Basic on M IM-Plus V2.3
Basic-Plus-2 (PDP11)

For example:

```
PIP MPG.B2S;4/DE
```

In this example, only version 4 of MPG.B2S is deleted.

To delete a file while using DCL, type:

```
DELETE filespec[;version]
```

where:

filespec	Is the name and type of the file you want to delete.
version	Is the version number of the file you want to delete. If you specify a version number, you delete only that version of the file. If you do not include the version number, DCL deletes the latest version of the file. If you use an asterisk (*) for the version number, you delete all versions of the file.

For example:

```
DELETE MPG.B2S;*
```

In this example, DCL deletes all versions of MPG.B2S.

1.4.7 Creating Executable Images

Section 1.3.1 describes how to create and run a BASIC program in the BASIC environment. While this method is convenient, it is not the most efficient way to execute programs. The most efficient way to execute programs is to create an executable image. To create an executable image of your program, you must perform the following steps:

1. Compile the program
2. Create overlay descriptor language (ODL) and command (CMD) files for the program
3. Link the program

After you create an executable image, you can run it with the CLI RUN command.

When you use this method to execute a program, you need to compile and link your program only once. In contrast, each time you run a program in the BASIC environment, the BASIC compiler must translate your program first and then execute it.

When you compile a BASIC program, the compiler creates a file called an *object module file*. In an object module file, each BASIC language element is replaced with the name of a routine from the BASIC Object Time System (OTS). These routines reside in libraries and are called threads. Each thread is a symbol that must be translated into executable code before your program can execute.

When you link your program, you use the Task Builder. The Task Builder links to libraries and replaces each thread name in the program with the address of the executable code that defines it. Then, the Task Builder places that code in an executable image file. When you execute your program, the system executes that image.

1.4.7.1 Compiling Programs

You compile your program in the BASIC environment by placing a copy of your program in memory and by using the COMPILE command.

The format of the COMPILE command is:

```
COMPILE [filename]
```

where:

filename	Is the name of the output file called an object module file. If you do not supply an output file name, the BASIC compiler generates an object module file with the same file name as your program and the default file type OBJ.

In the following example, the program named CREATE.B2S is created in the BASIC environment and then compiled. BASIC generates an object module file named CREATE.OBJ for this program.

```
BASIC2

NEW CREATE<RET>

BASIC2

1       MAP (ACCT) STRING ACCT.NAME=50%, &<RET>
<TAB>           ACCT.ACCOUNT=6%<RET>
<TAB>   !<RET>
 Open_file:     OPEN 'ACCT.DAT' FOR INPUT AS FILE #1, &<RET>
<TAB>           ORGANIZATION SEQUENTIAL FIXED, &<RET>
<TAB>           MAP ACCT<RET>
<TAB>   !<RET>
 Input_record:  LINPUT 'What is the account name'; ACCT.NAME<RET>
<TAB>   !<RET>
<TAB>           GOTO Done IF ACCT.NAME=' '<RET>
<TAB>           !<RET>
<TAB>           LINPUT 'What is the account number';ACCT.ACCOUNT<RET>
<TAB>           !<RET>
<TAB>           ! Write records<RET>
<TAB>           !<RET>
<TAB>           PUT #1%<RET>
<TAB>           !<RET>
 Done:          CLOSE #1%<RET>
<TAB>           END<RET>

COMPILE<RET>

BASIC2
```

If you want to compile a program that is not currently in memory, use the OLD command to place a copy of it in memory and then use the COMPILE command:

```
BASIC2
OLD CREATE<RET>
BASIC2
COMPILE<RET>
BASIC2
```

1.4.7.2 Linking and Running Programs

You use the Task Builder to link your program. The Task Builder is an RSX–11M/M–PLUS system program that uses libraries to translate your object module code into executable code. The Task Builder generates an executable task image for your program. This image is stored in a file with the same file name as your program and the default file type TSK.

The Task Builder requires a command (CMD) file and an overlay descriptor language (ODL) file to link a program. The Task Builder uses the CMD file for instructions on which libraries and ODL files to use, how many words to extend your task, and so on. The Task Builder uses the ODL file for instructions on how to overlay segments of your program and the libraries your program needs during the execution of your program. See the *RSX–11M/M–PLUS Task Builder Manual* for information on Task Builder command files and overlay descriptor language files.

You can use the BASIC BUILD command to generate these files for your program. For example, this command sequence generates ODL and CMD files for the program compiled in Section 1.4.7.1:

```
BASIC2
OLD CREATE<RET>
BASIC2
BUILD/SEQUENTIAL<RET>
BASIC2
```

This command sequence places a copy of CREATE.B2S into memory. The BUILD command generates a command file named CREATE.CMD and an overlay descriptor language file named CREATE.ODL.

Once you have compiled your program and created the CMD and ODL files for your program, return to the RSX–11M/M–PLUS system command level to use the Task Builder to link your program.

To invoke the Task Builder while using MCR, type:

```
TKB @filename
```

where:

filename — Is the name of the command file for your program.

To invoke the Task Builder while using DCL, type:

```
LINK/BASIC filename
```

where:

filename — Is the name of the command file for your program. The Task Builder searches for the file you name with a CMD file type.

For example:

```
LINK/BASIC CREATE
```

In this example, the DCL LINK command uses the Task Builder command file named CREATE.CMD to create an executable image file named CREATE.TSK.

Once the task image for your program has been created, you can execute it with the RUN command. For example:

```
RUN CREATE(RET)
What is the account name? Reilly(RET)
What is the account number? 123456(RET)
```

The RUN command runs the executable image for the program compiled in Section 1.4.7.1.

Chapter 2
BASIC Compiler Commands

The conditions in effect when you execute BASIC compiler commands, compile programs, or generate CMD and ODL files in the BASIC environment are called defaults. Your system manager sets the compiler defaults. However, you can change the defaults to suit your own needs.

You can specify different conditions in the BASIC environment by adding qualifiers to compiler commands. For example, you may want to specify that all floating-point numbers have 16 digits of precision (DOUBLE) rather than six (SINGLE). To do this, use the DOUBLE qualifier to the BASIC COMPILE command:

```
COMPILE/DOUBLE
```

This chapter explains the following:

- BASIC compiler defaults
- BASIC initialization files
- BASIC compiler commands

2.1 Compiler Defaults

You use the SHOW command to display current compiler defaults. BASIC displays the current default conditions for the BASIC environment on your operating system. If a NO precedes the setting, it is not in effect. Although the following display may look different from the display on your system, it gives you a sample of what to expect.

```
BASIC2

SHOW
PDP-11 BASIC-PLUS-2 V2.3-00 using FPU with run support
ENVIRONMENT INFORMATION:
    Current edit line : 0
    NO Modules loaded
    NO Main module loaded

DEFAULT DATA TYPE INFORMATION:
    Data type : REAL
    Real size : SINGLE
    Integer size : WORD
    Scale factor : 0
COMPILATION QUALIFIERS:
        Object
    NO Macro
        Lines
        Warnings
    NO Debug records
    NO Syntax checking
    Flag : Declining
    Variant : 0
RMS FILE ORGANIZATION:
    NO Index
    NO Relative
    NO Sequential
    NO Virtual
LISTING FILE INFORMATION:
    NO Source
    NO Cross Reference
    NO Keywords
        60 lines by 132 columns
BUILD QUALIFIERS:
    NO Dump
    NO Map
    NO Cluster
    NO I- and D-Space
    Task extend  : 512
    RMS ODL file : LB:RMSRLX
    BP2 Disk lib : LB:BP2OTS
    BP2 Resident lib : NONE
    RMS Resident lib : LB:RMSRES
```

This display tells you that the compiler uses floating-point hardware (FPU). The display also tells you that the compiler supports the RUN command; therefore, the compiler supports the LOAD command and immediate mode statements as well. The display also tells you the following information:

ENVIRONMENT INFORMATION

- *Current edit line* tells you the number of the line currently being edited. You use this information to keep track of the current edit line during an editing session using the EDIT command.
- *NO Modules loaded* tells you that no object modules are currently loaded in the BASIC environment.
- *NO Main module loaded* tells you that no main program module is loaded in the BASIC environment.

DEFAULT DATA TYPE INFORMATION

- *Data type :* tells you the data type of any variables not explicitly declared and not ending in a percent or dollar sign. In this case, the data type indicates that the relevant variables are real or floating-point variables. You can override the default variable data type by adding the TYPE_DEFAULT qualifier to the COMPILE or SET command.
- *Real size :* tells you the default precision for floating-point numbers. You can override the default precision for floating-point numbers by adding the SINGLE or DOUBLE qualifier to the COMPILE or SET command.
- *Integer size :* tells you the default precision for INTEGER numbers not explicitly data typed. You can override the default integer size by adding the BYTE, WORD, or LONG qualifier to the COMPILE or SET command.
- *Scale factor :* tells you the scale factor for scaled arithmetic. You can override the default scale factor by using the SCALE command.

COMPILATION QUALIFIERS

- *Object* tells you that the compiler generates an object module file for your program when you use the COMPILE command. You can override this default by using the /NOOBJECT qualifier to the COMPILE or SET command.
- *No Macro* tells you that the compiler does not generate a file containing the MACRO source code for your program when you use the COMPILE command. You can override this default by using the MACRO qualifier to the COMPILE or SET command.
- *Lines* tells BASIC to include line number information for your program so you can use RESUME with line number statements, ERL functions, or ERL debugger commands. Although you can save space and consequently speed program execution by specifying NOLINE, you receive an error message when you use the RESUME command with a line number statement, the ERL function, or the ERL debugger command. You can override the LINES default by adding the /NOLINE qualifier to the COMPILE or SET command.
- *Warnings* tells BASIC to display all compile-time errors, including errors of a "warning" severity. You can override this feature with the /NOWARNING qualifier to the COMPILE or SET command so that only errors of "error" or "fatal" severity are displayed at the terminal.
- *NO Debug records* tells BASIC not to provide information about your program for the BASIC debugger. You can override this default by adding the /DEBUG qualifier to the COMPILE or SET command. See Chapter 6 for more information on the BASIC debugger.
- *NO Syntax checking* tells BASIC not to check each program line after it is typed to make sure all BASIC elements are syntactically correct. Although syntax checking is a good learning tool, it does slow compiler response time. You can override this default by adding the /SYNTAX qualifier to the SET command.
- *Flag : Declining* tells the BASIC compiler to signal a message when you use language features that are not recommended for new program development. When the compiler finds such a feature, it displays the message "%Language feature is declining." You can override this default by adding the /FLAG:NODECLINING qualifier to the COMPILE or SET command.
- *Variant :* tells the BASIC compiler what variant value to use for the %VARIANT compiler directive. See the *BASIC User's Guide* for information on compiler directives.

RMS FILE ORGANIZATION

- *NO Index* tells the ODL file not to provide support for RMS indexed file operations. You can override this setting by adding the /IND qualifier to the BUILD command or by adding an ORGANIZATION INDEXED clause to the OPEN clause.
- *NO Relative* tells the ODL file not to provide support for RMS relative file operations. You can override this setting by adding the /REL qualifier to the BUILD command or by adding an ORGANIZATION RELATIVE clause to the OPEN clause.
- *NO Sequential* tells the ODL file not to provide support for RMS sequential file operations. You can override this setting by adding the /SEQ qualifier to the BUILD command or by adding an ORGANIZATION SEQUENTIAL clause to the OPEN clause.
- *NO Virtual* tells the ODL file not to provide support for RMS virtual array and block I/O file operations. You can override this setting by adding the /VIR qualifier to the BUILD or SET command, or by adding an ORGANIZATION VIRTUAL clause to the OPEN clause.

LISTING FILE INFORMATION

- *NO Source* tells BASIC not to include a copy of your program in the listing file. The listing file takes the name of your program and the default file type LST. You can override this default by adding the /LIST qualifier to the COMPILE or SET command.
- *NO Cross Reference* tells BASIC not to include cross-referencing information about the variables, arrays, function definitions, and so on in your program. This information can help you optimize your program by identifying duplicate operations. Cross-reference information is stored in the listing file for your program. You can override this default by adding the /CROSS_REFERENCE qualifier to the COMPILE or SET command.
- *NO Keywords* tells BASIC not to include additional cross-reference information about the BASIC keywords your program uses. You can override this default by adding the /CROSS_REFERENCE:KEYWORDS qualifier to the COMPILE or SET command.
- *60 lines by 132 columns* specifies the page size of the listing file. You can override this default by adding the /PAGE_SIZE:qualifier, the /WIDTH: qualifier, or both qualifiers to the COMPILE or SET command.

BUILD QUALIFIERS

- *NO Dump* tells BASIC not to generate a post-mortem dump for your program if it aborts with a fatal error. Although a post-mortem dump helps speed Software Performance Report (SPR) response time, it slows compile time. You can override this default by adding the /DUMP qualifier to the BUILD command.
- *NO Map* tells the Task Builder not to generate a file of the memory allocation map for your program. This information is stored in a file with the same name as your program and the default file type MAP. You can override this default by adding the /MAP qualifier to the COMPILE or SET command.
- *NO Cluster* tells the Task Builder not to cluster the default BASIC–PLUS–2 and RMS–11 resident libraries. You can override this default by adding the /CLUSTER qualifier to the BUILD or SET command. See Section 7.6 for more information on clustering.
- *NO I- and D-Space* tells the Task Builder not to use I- and D-Space. You can override this setting by adding the /IDS qualifier to either the BUILD or SET command. See Chapter 3 for more information.
- *Task extend :* tells the Task Builder how many bytes to extend the memory allocation for your program. The default extension is 512 bytes. You can override this default by adding the /EXTEND qualifier to the BUILD or SET command.
- *RMS ODL file :* tells the Task Builder which RMS ODL file to use to overlay segments of the RMS–11 object module library. You can override the default by adding the /ODLRMS qualifier to the BUILD command or by using the ODLRMS command.
- *BP2 Disk lib :* tells the Task Builder which disk-resident, object module library to use to link your program. You can override the default by adding the /DSKLIB qualifier to the BUILD command or by using the DSKLIB command.
- *BP2 Resident lib :* tells the Task Builder which BASIC memory-resident library to use to link your program. You can override the default by adding the /BRLRES or /LIBRARY qualifier to the BUILD command or by using the BRLRES or LIBRARY command.

- *RMS Resident lib :* tells the Task Builder which RMS–11 memory-resident library to use to link your program. The default is to use RMSRES. You can override the default by adding the /RMSRES qualifier to the BUILD command or by using the RMSRES command.

2.1.1 Changing Compiler Defaults

You can change BASIC compiler defaults by adding qualifiers to the BUILD, COMPILE, or SET command and by using certain compiler commands.

When you change a default by using qualifiers to the BUILD or COMPILE command, the new default remains in effect for only one compile operation. For example:

```
COMPILE/DOUBLE/LIST/NOOBJECT
```

This command line tells the BASIC compiler to do the following:

- Specify that all floating-point numbers not explicitly data typed are DOUBLE precision
- Write the program to a disk file
- Compile the program without generating an object module file

When you change a default using qualifiers to the SET command, the new default remains in effect until you (1) specify another default with the SET command, (2) override the default by adding qualifiers to the BUILD or COMPILE command, (3) set a new default with a compiler command, or (4) leave the BASIC environment.

In the following example, the SET commands specify the same defaults as those specified with the COMPILE command in the previous example:

```
BASIC2
SET /DOUBLE
BASIC2
SET /LIST
BASIC2
SET /NOOBJECT
```

You can set defaults automatically with an initialization file. This method is explained in the following section.

2.2 Setting Compiler Defaults with an Initialization File

Using compiler commands and qualifiers in the BASIC environment is useful when the new defaults affect only a few compilations. However, if you use a particular set of defaults frequently, changing the defaults each time you enter the BASIC environment can be tedious. You can have certain defaults take effect automatically whenever you enter the BASIC environment by using an *initialization file*.

When you invoke BASIC, it searches for two initialization files named BP2INI.nnn (where nnn is the 3-character name you use to invoke BASIC). BASIC searches for the first file in the system account, LB:[1,2], and then searches in your current default directory for the second file.

The system manager sets defaults for all users on the system by creating a BP2INI file in LB:[1,2]. Each time the BASIC compiler is invoked, it executes the commands in the system BP2INI file.

Similarly, you can define your own compiler defaults by creating a BP2INI file in your default account. If either or both BP2INI files exist, BASIC executes the compiler commands the files contain when you enter the BASIC environment.

Note that BASIC executes the system initialization file first. In addition, the commands in the files are performed in sequential order. Therefore, if a command is repeated or two commands contradict each other (such as SET DOUBLE and SET SINGLE), the commands in your private initialization file take precedence over the commands in the system initialization file.

2.2.1 Creating a BASIC Initialization File

You use a text editor to create your initialization file. The file can contain any of a subset of BASIC compiler commands. Each command must appear on a separate line. Comments are not allowed; however, you can use blank lines to separate the commands for readability.

The following list describes the allowable BASIC commands and their functions:

- **Setting COMPILE and BUILD qualifiers**

  You can use the LOCK and SET commands to define COMPILE and BUILD defaults. Any valid SET qualifier can be included in the initialization file.

- **Changing the default BASIC and RMS libraries and RMS ODL files**

  You can use all of the BASIC compiler commands that specify default object module or memory-resident libraries. That is, you can use the BRLRES, DSKLIB, LIBRARY, and RMSRES commands. You can also use the ODLRMS command to specify a new default RMS ODL file.

  However, in the initialization file, these commands cannot prompt for library names or ODL file specifications. Therefore, if an initialization file contains a command without the necessary library name or file specification, BASIC reports the error "no file specified for command in initialization file."

- **Changing the scale factor**

  You can use the SCALE command to specify the scale factor in an initialization file.

- **Displaying the current defaults**

  The SHOW command is a valid command in an initialization file. You can use the SHOW command to display the current compiler defaults as you enter the BASIC environment.

  If you use the SHOW command in your initialization file, make sure it is the last command in the file. If it is not, later commands can change the defaults, so that the defaults the SHOW command displays as you enter the environment will be misleading.

2.2.2 Messages Reported from the Initialization Files

BASIC does not display the commands in the initialization file as it executes them. BASIC also does not report when it cannot find either of the initialization files.

However, if either of the initialization files contains an invalid command or a command with an invalid argument, BASIC reports the appropriate compile-time error with additional text indicating that the error occurred in the initialization file. For example, if one of the initialization files contains the SET command with an invalid qualifier, such as SET/ABC, BASIC reports the error "illegal switch usage–ABC from initialization file."

2.3 Compiler Commands

The COMPILE, BUILD, and SET commands are all *compiler commands*. Compiler commands (or BASIC environment commands) are similar to CLI commands in the RSX–11M/M–PLUS system command level: they perform functions to assist you in your primary task of creating and executing BASIC programs. Compiler commands let you do the following:

- Display, edit, and merge BASIC programs. These commands include:

  APPEND
  DELETE
  EDIT
  LIST
  NEW
  RENAME
  SCRATCH
  SEQUENCE

- Move BASIC programs to and from storage. These commands include:

  OLD
  REPLACE
  SAVE
  UNSAVE

- Execute programs. These commands include:

  CONTINUE
  LOAD
  RUN

- Display online documentation. These commands include:

  HELP
  IDENTIFY
  INQUIRE
  SHOW

- Set specific defaults. These commands include:

  BRLRES
  BUILD
  COMPILE
  DSKLIB
  LIBRARY
  ODLRMS
  RMSRES
  SCALE
  SET

Compiler commands can be abbreviated to three letters and cannot be preceded by numbers or spaces. Table 2–1 shows a list of compiler commands and their functions, and the sections that follow describe these commands in detail.

Table 2–1: BASIC Compiler Commands

Command	Function
APPEND	Merges the program you specify with the program currently in memory.
BRLRES	Selects a BASIC memory-resident library. See Chapter 7 for information on the BRLRES command.
BUILD	Generates a command (CMD) file and overlay descriptor language (ODL) file for the Task Builder.
$ Command	Allows execution of RSX–11M/M–PLUS system commands from the BASIC environment.
COMPILE	Generates an object module for a BASIC program compiled in the BASIC environment.
CONTINUE	Resumes execution after a STOP statement or a CTRL/C.
DELETE	Erases one or more specified lines from a BASIC program.
DSKLIB	Selects a BASIC object module library. See Chapter 7 for information on the DSKLIB command.
EDIT	Changes text within a program line.
EXIT	Returns to RSX–11M/M–PLUS system command level.
HELP	Displays information on BASIC language elements.
IDENTIFY	Causes BASIC to display an identification header.
INQUIRE	Is identical to the HELP command.
LIBRARY	Selects a BASIC memory-resident library. See Chapter 7 for information on the LIBRARY command.
LIST	Displays the program currently in memory.
LISTNH	Is identical to the LIST command but does not display an identification header.
LOAD	Loads an object module into memory.
LOCK	Is identical to the SET command.
NEW	Clears memory for the creation of a new program.
ODLRMS	Selects an RMS–11 overlay descriptor language (ODL) file. See Chapter 7 for information on the ODLRMS command.

(continued on next page)

Table 2–1 (Cont.): BASIC Compiler Commands

Command	Function
OLD	Copies and places a specified BASIC program into memory.
RENAME	Changes the name of the program currently in memory.
REPLACE	Replaces a stored program with the program currently in memory.
RMSRES	Selects an RMS–11 memory-resident library. See Chapter 7 for information on the RMSRES command.
RUN	Executes the program currently in memory or a specified BASIC program.
RUNNH	Is identical to RUN but does not display an identification header.
SAVE	Creates a copy of the program currently in memory and writes it to the default device or a specified device.
SCALE	Controls accumulated round-off errors for numeric operations.
SCRATCH	Clears memory.
SEQUENCE	Generates line numbers for program lines.
SET	Specifies compiler defaults.
SHOW	Displays the current compiler defaults.
UNSAVE	Deletes a specified file.

2.3.1 APPEND

The APPEND command combines the BASIC program you specify with the program currently in memory. Its format is:

```
APPEND [ file-spec ]
```

where:

file-spec Is the program to be appended.

The program in memory must be a BASIC program either (1) placed in memory with the OLD command or (2) created in the BASIC environment. If both programs contain a line with the same number, the appended program line replaces the current program line.

If you type APPEND without specifying a file name, BASIC prompts you as follows:

```
Append file name -
```

Respond with a file name. If you respond by typing a carriage return, BASIC searches for a file named NONAME with the default file type B2S. If the compiler cannot find the file, it displays the error message "?Can't find file or account."

This example displays the program to be appended (PROD), creates a new program named MPG, and then merges PROD with MPG:

```
BASIC2

OLD PROD

BASIC2

LISTNH

100       PRINT 'This program displays the product of two numbers.'
200       INPUT 'Enter two numbers and separate them with a comma';A,B
300       PRINT 'Their product is ';A * B
400       END

BASIC2

NEW MPG
10        INPUT 'What is your total mileage';MIL
20        INPUT 'What is the number of gallons';GAL
30        MPG = MIL/GAL
100       PRINT 'Your car is getting ';MPG;' miles per gallon.'
200       END
APPEND PROD

BASIC2

LISTNH
10        INPUT 'What is your total mileage';MIL
20        INPUT 'What is the number of gallons';GAL
30        MPG = MIL/GAL
100       PRINT 'This program displays the product of two numbers.'
200       INPUT 'Enter two numbers and separate them with a comma';A,B
300       PRINT 'Their product is ';A * B
400       END

BASIC2
```

The APPEND command does not change the name of the program in memory.

2.3.2 BUILD

The BUILD command generates a command (CMD) file and an overlay descriptor language (ODL) file for the Task Builder. The CMD file contains instructions that enable the Task Builder to link your program module or modules using libraries and other routines. The ODL file specifies how segments of your program are overlaid when you run it. The format of the BUILD command is:

BUILD [prog-nam [, sub-nam,...]] [/qualifier ...]

where:

prog-nam — Is the file name of the main program. This name becomes the file name for the CMD and ODL files.

sub-nam — Is the name of a subprogram to be included in the task image. See Chapter 5 for information on subprograms.

/qualifier — Specifies additional instructions for the compiler. If you precede a qualifier with NO, it is not in effect. The defaults for these qualifiers are selected when BASIC is installed. See Table 2–2 for a complete list of BUILD qualifiers.

For example:

```
BASIC2
BUILD MAIN
```

In this example, the BUILD command creates two files: MAIN.CMD and MAIN.ODL. The Task Builder uses these files to control the creation of an executable task image.

Table 2–2 lists the BUILD command qualifiers.

Table 2–2: BUILD Command Qualifiers

Qualifier	Function
BRLRES: { lib-nam / NONE / file-spec }	Lets you specify a BASIC or user-created memory-resident library to be used by the Task Builder. Lib-nam is the name of a memory-resident library supplied by BASIC. NONE tells the Task Builder not to link your task to any BASIC memory-resident library. File-spec is a user-created memory-resident library. See Chapter 7 for more information.
[NO]CLUSTER	Tells the Task Builder to cluster the default BASIC–PLUS–2 and RMS–11 resident libraries.
DSKLIB: { file-spec / lib-nam }	Lets you specify a disk-resident, object module library to be used by the Task Builder. Lib-nam is the name of a library supplied by BASIC (BP2OTS.OLB). File-spec is a user-created object module library. See Chapter 7 for more information.
[NO]DUMP	Tells the Task Builder to generate a post-mortem dump for a program if it aborts with a fatal error.
EXTEND:int-const	Specifies the minimum increment used when the Task Builder extends the memory allocation for your task. The Task Builder rounds this value up to the nearest 32-word boundary. The maximum extension is limited by your operating system. See the *RSX–11M/M–PLUS Task Builder Manual* for information.
[NO]IDS	Causes the Task Builder to build the task in I-Space and D-Space. See Chapter 3 for more information.

(continued on next page)

Table 2–2 (Cont.): BUILD Command Qualifiers

Qualifier	Function
[NO]IND	Causes the Task Builder to include the code needed for indexed file operations. BASIC sets this qualifier automatically when you compile a program containing an ORGANIZATION INDEXED clause in the OPEN statement. See the *BASIC User's Guide* for more information.
LIBRARY: lib-nam NONE file-spec	Lets you specify a BASIC or user-created memory-resident library to be used by the Task Builder. Lib-nam is the name of one of the memory-resident libraries supplied by BASIC. NONE tells the Task Builder not to link your program to any BASIC memory-resident library. File-spec is a user-created memory-resident library. See Chapter 7 for more information.
[NO]MAP	The MAP qualifier causes the Task Builder to generate a file that shows how memory is allocated for your task. This file uses the same name as the program you link and the default file type MAP. See the *RSX–11M/M–PLUS Task Builder Manual* for more information.
ODLRMS: odl-nam file-spec NONE	Specifies the ODL file for RMS–11 libraries. Odl-nam is an ODL file supplied by RMS–11. File-spec is a user-created ODL file. You must use the full file specification for a user-created ODL file. NONE tells the Task Builder not to use any RMS–11 ODL file. See Chapter 7 for more information.
[NO]REL	Causes the Task Builder to include the code needed for relative file operations. BASIC sets this qualifier automatically when you compile a program containing an ORGANIZATION RELATIVE clause in the OPEN statement. See the *BASIC User's Guide* for more information.
RMSRES: lib-nam NONE file-spec	Lets you specify an RMS–11 or user-created memory-resident library to be used by the Task Builder. Lib-nam is the name of a memory-resident library supplied by RMS–11. NONE tells the Task Builder not to link your task to any RMS–11 library. File-spec is a user-created RMS–11 memory-resident library. See Chapter 7 for more information.
[NO]SEQ	Causes the Task Builder to include the RMS–11 code needed for sequential file operations. BASIC sets this qualifier automatically when you compile a program containing an ORGANIZATION SEQUENTIAL clause in the OPEN statement. See the *BASIC User's Guide* for more information.
VIR	Causes the Task Builder to include the RMS–11 code needed for virtual array and block I/O file operations. BASIC sets this qualifier automatically when you compile a program containing an ORGANIZATION VIRTUAL clause in the OPEN statement. See the *BASIC User's Guide* for more information.

In the following example, the BUILD command generates an ODL and a CMD file that support SEQUENTIAL file organization and tells the Task Builder to use the RMS–11 resident library named RMSRES:

```
BASIC2

OLD MYPROG

BASIC2

BUILD/SEQ/RMSRES:RMSSEQ

BASIC2
```

2.3.3 $ Command

You can execute an RSX–11M/M–PLUS system command while in the BASIC environment by preceding it with a dollar sign ($). Its format is:

```
$ system-command
```

where:

system-command	Is an RSX–11M/M–PLUS system command. See the *RSX–11M/M–PLUS MCR Operations Manual* and the *RSX–11M/M–PLUS Command Language Manual* for information on system commands.

In this example, BASIC passes the PRINT command to the operating system for execution and then returns to the BASIC environment:

```
BASIC2

$ PRINT MYPROG.B2S

PRI - Job 22, name "MYPROG", submitted to the queue "PRINT "

BASIC2
```

The context of the BASIC environment and the program currently in memory do not change after the operating system executes its command and returns control to the BASIC compiler.

2.3.4 COMPILE

The COMPILE command creates an object module for the program currently in memory. Its format is:

```
COMPILE [file-nam] [/qualifier] ,...
```

where:

file-nam — Specifies a name for the output file.

/qualifier — Specifies a COMPILE qualifier. You can abbreviate all COMPILE qualifiers to three letters. If you precede a qualifier with NO, it is not in effect. The default for these qualifiers are selected when BASIC is installed.

Table 2–3 lists the COMPILE command qualifiers.

Table 2–3: COMPILE Command Qualifiers

Qualifier	Function
BYTE	Causes the compiler to allocate eight bits of storage to all integer data not explicitly typed in the program. If BYTE is in effect, any integer values not explicitly typed are treated as BYTE values and must be in the range –128 to +127 or BASIC displays the error message "Integer overflow." See the *BASIC User's Guide* for more information.
[NO]CHAIN	Enables programs to use another program when you use the CHAIN statement. If the program has more than 200 line numbers, NOCHAIN reduces the memory needs of the output program by disabling the storage of line numbers in memory. See the *BASIC Reference Manual* for more information.
[NO]CROSS_REFERENCE[:[NO]KEYWORDS]	Generates cross-reference information about program variables, arrays, function definitions, and so on for the program being compiled and places this information in the listing file. Cross-reference information can help you optimize your programs by identifying duplicate operations. CROSS_REFERENCE:KEYWORDS generates additional cross-reference information about BASIC keywords for the program being compiled and places this information in the listing file.
[NO]DEBUG	Provides information for the BASIC debugger. The debugger is a tool that helps you debug your programs. If DEBUG is in effect, BASIC automatically sets the LINE default. See Chapter 6 for more information on the debugger.
[NO]FLAG:[NO]DECLINING	Causes BASIC to warn you when your programs use BASIC features that are not recommended for new program development. If your program uses such a feature and you use this qualifier, BASIC displays the error message "%Language feature is declining."
DOUBLE	Causes the compiler to allocate 64 bits of storage in double-precision format for all floating-point data not explicitly typed in the program. If the DOUBLE default is in effect, all floating-point values not explicitly typed are treated as double-precision values and must be in the range $2.9 * 10^{-39}$ to $1.7 * 10^{38}$ or BASIC displays the error message "Floating-point overflow." See the *BASIC User's Guide* for more information.

(continued on next page)

Table 2–3 (Cont.): COMPILE Command Qualifiers

Qualifier	Function
[NO]LINE	Includes line number information in object modules. If you specify NOLINE in a program containing a RESUME statement without a line number, BASIC sets the LINE default in effect and displays the error message "RESUME overrides NOLINE." If NOLINE is in effect and you use the DEBUG qualifier or the ERL function, BASIC resets the qualifier to LINE.
[NO]LIST	Tells BASIC to write a copy of the program currently in memory to a file on disk. The listing file takes the file name of the program and the default file type LST.
LONG	Causes the compiler to allocate 32 bits of storage to all integer data not explicitly typed in the program. If LONG is in effect, all untyped integer values are treated as LONG values and must be in the range –2147483647 to 2147483647 or BASIC displays the error message "Integer overflow." See the *BASIC User's Guide* for more information.
[NO]MACRO	Translates your program into MACRO code and saves it in a file that takes the file name of your program and the default file type MAC. The file generated by COMPILE/MACRO command can be assembled.
[NO]OBJECT	Generates an object module file that takes the file name of the program and the default file type OBJ. Use the NOOBJECT qualifier to check your program for errors without creating an object module file.
PAGE_SIZE:int-const	Allows you to select the page length of the listing file. Int-const must be in the range from 1 to 60.
SINGLE	Causes the compiler to allocate 32 bits of storage in single-precision format for all floating-point data not explicitly typed in the program. If SINGLE is in effect, all floating-point values not explicitly typed are treated as single precision values and must be in the range $2.9 * 10^{-39}$ to $1.7 * 10^{38}$ or BASIC displays the error message "Floating-point overflow." See the *BASIC User's Guide* for more information.
TYPE.DEFAULT: EXPLICIT / REAL / INTEGER	Tells the compiler that all variables not explicitly declared in your program are of the data type (REAL or INTEGER) you specify. EXPLICIT means that all variables in the program must be explicitly declared. See the *BASIC User's Guide* for more information.
VARIANT:int-const	Establishes int-const as the value to be used for compiler directives. You can reference the variant value in a lexical expression using the lexical function %VARIANT. Int-const always uses the data type WORD. See the *BASIC User's Guide* for more information.
[NO]WARNING	Tells BASIC to display all compile-time errors, including errors of a "warning" severity.

(continued on next page)

Table 2–3 (Cont.): COMPILE Command Qualifiers

Qualifier	Function
WIDTH:int-const	Lets you select the width of the page for the listing file. Int-const must be in the range from 72 to 132.
WORD	Causes the compiler to allocate 16 bits of storage to all integer data not explicitly typed in the program. If WORD is in effect, all integer values not explicitly typed are treated as WORD values and must be in the range –32767 to 32767 or BASIC displays the error message "Integer overflow." See the *BASIC User's Guide* for more information.

2.3.5 CONTINUE

The CONTINUE command resumes program execution after BASIC executes a STOP statement or after you press CTRL/C. Its format is:

```
CONTINUE
```

2.3.6 DELETE

The DELETE command removes a line or lines you specify from the BASIC program currently in memory. Its format is:

```
DELETE lin-num [ sep lin-num] ,...
```

where:

lin-num	Is a program line number.
sep	Is a comma or a hyphen.

If you separate line numbers with commas, BASIC removes only those program lines. If you separate line numbers with a hyphen (-), BASIC removes those program lines and all program lines between them. For example:

Command	Meaning
DELETE 10	Removes line 10 from the program.
DELETE 50, 100	Removes lines 50 and 100 from the program.
DELETE 50, 100–190	Removes line 50 and lines 100 through 190 from the program.
DELETE 1000-	Removes all lines from 1000 to the end of the program.

If you do not specify a line number, DELETE has no effect.

2.3.7 EDIT

The EDIT command lets you edit the BASIC program currently in memory. If you want to change text in a single program line, the EDIT format is:

EDIT [[lin-num [–lin-num]] search-clause [replace-clause]]

search-clause: delim unq-str1 delim

replace-clause: [unq-str2] delim [int-const1]

where:

lin-num	Is the line to be edited. If you do not specify a line number, the default is the last edited line. If there is no previously edited line and you do not specify a line number, the EDIT command causes you to enter editing mode. If you forget which program line is the last edited line, type the SHOW command to display the current edit line.
delim	Can be any printing character not used in the search or replacement string.
search-clause	Is the string you want to remove. If you do not specify a search string, BASIC displays the line but does not delete it. Unq-str1 is the search string you want to remove or replace.
replace-clause	Is the string to be substituted for the search string. If you do not specify a replacement string, BASIC deletes the search string. Unq-str2 is the replacement string.
int-const	Is a number specifying which occurrence of unq-str1 you want to replace. The default value for int-const is the first occurrence of unq-str1.

This example replaces the first occurrence of "LEFT$" with "RIGHT$" in line 100:

```
EDIT 100 /LEFT$/RIGHT$/
```

This example displays line 2000. Line 2000 becomes the current edit line if you do not specify a line number in subsequent EDIT commands:

```
EDIT 2000
```

This example starts the search on the third text line of program line 30 and replaces the first occurrence of "LEFT$" with "RIGHT$":

```
EDIT 30 /LEFT$/RIGHT$/3
```

This example removes the second occurrence of "LEFT$(" from line 300. Note that you must specify delimiters around the null replacement string. Otherwise, the EDIT command replaces the first occurrence of "LEFT$(" with "2".

```
EDIT 300/LEFT$(//2
```

If you type the EDIT command with no argument, BASIC enters editing mode and displays its command prompt (*). Once in editing mode, you can use any editing mode subcommands listed in Table 2–4.

Table 2–4: Editing Subcommands

Editing Subcommand	Function
D[EFINE] str-exp	Allows you to create a macro definition. The macro consists of a series of editing mode commands in the order in which they are to execute. You can use any editing subcommands except DEFINE and EXECUTE to make up a definition. To begin a definition, enter the DEFINE command and the RETURN key, then enter the definition, line by line. When you have finished entering your macro definition, enter a carriage return or press CTRL/Z. Then BASIC returns to the editing mode prompt (*).
EXECUTE int-const	Executes the macro definition you created using the DEFINE command as many times as you specify.
E[XIT] or CTRL/Z	Allows you to exit from editing mode and return to the BASIC environment.
F[IND] str-exp	Searches the current program from the current edit line to the end of the program for the string you specify. Enter a carriage return to execute this subcommand.
I[NSERT] int-const	Allows you to add program lines after the current edit line or the line number you specify. Enter a carriage return to receive the prompt (>) to enter the program lines to be inserted. Enter the EXIT command to complete the insertion.
S[UBSTITUTE] str-exp	Performs the same function and accepts the same text parameters as the EDIT command for a single line. However, you cannot specify line numbers or line number ranges. Enter a carriage return to execute this command.

This command sequence enters editing mode. The DEFINE subcommand is used to define a macro definition where "REM" is found and then replaced with an exclamation point (!). The EXECUTE subcommand executes this macro once.

```
BASIC2

EDIT
*%No current edit line.
*DEFINE
Enter command sequence:
>FIND REM
>SUBSTITUTE /REM/!/
>EXIT
*EXECUTE
```

BASIC responds with:

```
10 REM This program balances your checkbook

"REM" found on line 10
*SUBSTITUTE 1
"REM" ( 1 ) replaced by "!".
1 substitution
```

This example uses the INSERT subcommand to enter the program line "200 VAR1 * INT2" after the line numbered 100:

```
*INSERT 100
Enter lines to be added after line 100
>200 VAR1 * INT2
>EXIT
*
```

2.3.8 EXIT

The EXIT command clears memory and returns control to the RSX–11M/M–PLUS system command level. Its format is:

EXIT

If you modify a program and issue the EXIT command before you copy it to disk with the SAVE or REPLACE command, BASIC displays "Unsaved Change has been made, CTRL/Z or EXIT to exit." This message warns you that the edits you made will be lost if you do not save the program by using the SAVE or REPLACE command. You can then save the program or retype the EXIT command (or press CTRL/Z) to exit from BASIC.

2.3.9 HELP

The HELP command lets you display information about BASIC on the terminal. Its format is:

HELP [unq-str] ...

where:

unq-str Is a BASIC keyword, command, or concept.

If you type HELP with no arguments, BASIC displays a list of main topics. If you already know the main topic for a particular subtopic, you can request help on that subtopic directly. For example:

```
HELP STATEMENTS MAT
```

This example causes BASIC to print (1) an explanation of the MAT statement, (2) its general format, and (3) an example of its use. When the keyword you enter is associated with other keywords, BASIC also lists the additional keywords. For example, the previous HELP statement displays the following final information:

```
Additional information available on:

INPUT     LINPUT     PRINT     READ
STATEMENTS MAT Subtopic?
```

To exit from HELP, press the return key until your terminal displays the BASIC prompt.

2.3.10 IDENTIFY

The IDENTIFY command prints an identification header containing the BASIC compiler name and version number. Its format is:

```
IDENTIFY
```

The following example illustrates the use of the IDENTIFY command on RSX–11M–PLUS systems:

```
BASIC2

IDENTIFY

PDP-11 BASIC-PLUS-2 V2.3-00
BASIC2
```

2.3.11 INQUIRE

The INQUIRE command is identical to the HELP command.

2.3.12 LIST and LISTNH

The LIST and LISTNH commands display one or more specified lines in a program. If you type LIST without specifying line numbers, BASIC displays a copy of the entire program that is currently in memory, in ascending line-number order. Its format is:

```
LIST[NH] [ [–] lin-num ] [ sep [ lin-num ] ]...
```

where:

lin-num	Is the number of the program line you want to display.
sep	Is a comma or hyphen.

Ths LIST command causes BASIC to display an identification header consisting of the program name, current time, and date before displaying the lines you specify. The LISTNH command suppresses the identification header and displays only the lines you specify. For example:

Command	Meaning
LIST 10	Displays header information and line 10.
LISTNH 50, 100	Displays lines 50 and 100.
LIST 50, 90, 100–190	Displays header information and lines 50, 90, and 100 through 190.

Note

In addition to LIST and LISTNH, you can use LIS and LISNH. BASIC accepts variations from COMNH to COMPILENH.

2.3.13 LOAD

The LOAD command makes an object module available for execution with the RUN command. You can use the LOAD command to make the object modules for each subprogram the main program calls available for execution with the RUN command. Its format is:

LOAD file-spec [+ file-spec] ...

where:

file-spec — Is the file name of an object module previously generated using the COMPILE command in the BASIC environment.

You can load only those object module files created by the BASIC compiler. If you do not specify a file name, the LOAD command removes all previously loaded object modules from memory.

The LOAD command accepts multiple file specifications. You separate multiple files with a plus sign. For example:

```
BASIC2

LOAD TEST1+TEST2

BASIC2
```

The object files in this command sequence are not executed until you issue the RUN command. Therefore, run-time errors in the loaded modules are not detected until you actually execute the program.

Each device and directory you specify applies to all subsequent file specifications until you specify a new directory or device. For example:

```
LOAD DB1:[200,30]PROG3_[200,35]PROG4_DB2:PROG5
```

This command loads three object files: (1) PROG3 from [200,30] on DB1:, (2) PROG4 from [200,35] on DB1:, and (3) PROG5 from [200,35] on DB2:.

The maximum number of object modules you can specify depends on the size of your programs. If they are too large (requiring over 32K words of virtual address space), they cannot be loaded or executed in the BASIC environment.

2.3.14 LOCK

The LOCK command lets you change compiler defaults. It is equivalent to the SET command. Its format is:

```
LOCK/qualifier[,/qualifier...]
```

where:

/qualifier	Is any qualifier to the SET command. See Section 2.3.24 for qualifiers to the SET command.

For example:

```
BASIC2
LOCK/DOUBLE
BASIC2
```

In this example, the LOCK command specifies that all subsequent compilations unless explicitly declared use double-precision floating-point numbers.

2.3.15 NEW

The NEW command clears memory and assigns a name to the program you want to create. Its format is:

```
NEW [ prog-nam ]
```

where:

prog-nam	Is the name of the program you want to create. The file name can be up to nine uppercase characters.

For example:

```
BASIC2
NEW MYPROG
BASIC2
```

This command assigns the file name MYPROG.B2S to the program. If you do not specify a name, BASIC prompts you as follows:

```
New file name -
```

Respond with a file name. If you press a RETURN key in response to the "New file name – " prompt, BASIC assigns the name NONAME with the default file type B2S.

2.3.16 OLD

The OLD command places a copy of a previously created file containing a BASIC program into memory. Its format is:

OLD [file-spec]

where:

file-spec	Is the file name of the program you want BASIC to make a copy of and place in memory.

For example:

```
BASIC2
OLD MYPROG
BASIC2
```

In this example, the OLD command copies MYPROG.B2S into memory.

If you do not specify a file name, BASIC prompts you as follows:

```
Old file name -
```

Respond with a file name. If you do not specify a file type, BASIC searches for a file with the name you specify and the default file type B2S. The default file name is NONAME. Thus, if you press a RETURN key in response to the "Old file name – " prompt, BASIC searches for NONAME.B2S.

2.3.17 RENAME

The RENAME command assigns a new name to the program currently in memory. Its format is:

RENAME [prog-nam]

where:

prog-nam	Is the new name of the program.

The following command sequence brings a program named PROG1.B2S into memory and changes its name to PROG2.B2S:

```
BASIC2
OLD PROG1
BASIC2
```

(continued on next page)

```
RENAME PROG2
BASIC2
```

In this example, the disk file from which PROG1 was read remains unchanged.

2.3.18 REPLACE

The REPLACE command writes the program in memory to a specified device. Its format is:

```
REPLACE [file-spec]
```

where:

file-spec	Is the name of the program to be written to disk.

A REPLACE command writes a copy of the program currently in memory to disk. If a file of the same name already exists, BASIC supersedes the old version. If you specify a file name with the REPLACE command, the program currently in memory is written to disk with the file name you specify.

2.3.19 RUN and RUNNH

The RUN command executes a program. This program can be (1) the program currently in memory, (2) object module(s) placed in memory with the LOAD command, (3) a combination of 1 and 2, or (4) the BASIC program you specify. You can use the RUN command to execute any BASIC program that is small enough to fit in memory without being overlaid. The format of the RUN command is:

```
RUN[NH] [ file-spec ] [ /qualifier(s) ]
```

where:

file-spec	Is the file name of the program you want to execute.
/qualifier(s)	Can be any COMPILE command qualifier except for the NOCROSS, NOLIST, NOMACRO, and NOOBJECT defaults that are always in effect.

If you do not supply a file specification, BASIC executes the program currently in memory:

```
BASIC2

OLD
Old file name--PROG1
BASIC2

RUN
```

In this example, the RUN command compiles and executes PROG1.

The RUN command displays an identification header consisting of the program name, the current date, and the current time before executing the program.

The RUN command does not create an object module file or a listing file. RUN executes your program under the current default conditions.

The RUNNH command is identical to the RUN command except it executes a program without displaying the identification header.

2.3.20 SAVE

The SAVE command copies the BASIC program currently in memory and writes it to a disk file. Its format is:

```
SAVE [file-spec]
```

where:

file-spec Is the file name of the program currently in memory that you want to save.

For example, if you enter the following program, a SAVE command causes BASIC to arrange the program in ascending line-number order and copy it to the disk DB0: with the file name TEST and the default file type B2S:

```
30      PRINT 'This is a test'
10      ! This is a test
SAVE DB0:TEST

BASIC2
```

This example specifies a storage device, a file name, and a file type:

```
SAVE DB1:NEWTST.B2BB
```

BASIC saves the file on disk DB1: with a file name of NEWTST and the file type BBB.

If the program in memory has no name and you issue the SAVE command with no argument, BASIC writes the program to your default disk device with the file name NONAME and the default file type B2S.

If you try to save a program that already exists, BASIC signals the error message "?File exists-RENAME/REPLACE." Use either the RENAME or REPLACE command to save a program that already exists.

2.3.21 SCALE

The SCALE command can overcome accumulated round-off errors by multiplying double-precision floating-point values by 10 raised to the specified scale factor before storing them. Its format is:

```
SCALE int-const
```

where:

int-const — Specifies the power of 10 you want to use as the scaling factor. It must be an integer value in the range from zero to six, inclusive.

If you do not supply an integer value, the current default scale factor is displayed:

```
BASIC2

SCALE
Current scale factor is  0

BASIC2
```

In this example, the value of scaled arithmetic is two. When a program is compiled, all floating-point numbers are either multiplied by 100 or divided by 100, where required:

```
BASIC2

SCALE 2
Scale factor has been set to  2

BASIC2
```

The scale factor you specify remains in effect until you specify a different scale factor with the SCALE command or you leave the BASIC environment, at which point the SCALE factor returns to its original default setting.

2.3.22 SCRATCH

The SCRATCH command clears memory of any programs, program lines, or immediate mode statements and removes any object module files loaded with the LOAD command. Its format is:

```
SCRATCH
```

If you edit a program in memory and then use the SCRATCH command, the edited version of the program is lost.

2.3.23 SEQUENCE

The SEQUENCE command automatically generates line numbers for program lines. The format of the SEQUENCE command is:

```
SEQUENCE [ lin-num ] [ ,int-const]
```

where:

lin-num — Is the first line number to be entered. If you do not specify a starting line number, the default is the last line number inserted by the SEQUENCE command. If there was no previous SEQUENCE command, the default is 100. If you specify a starting line number that already contains a statement, BASIC displays the error message "Attempt to sequence over existing statement" and returns to normal input mode.

int-const Is the number added to the previous line number to get the new line number. The default is 10.

After executing a SEQUENCE command, BASIC prompts with a new line number when you end each program line with a carriage return:

```
BASIC2

SEQUENCE 10,10
10    PRINT 'HELLO'
20

BASIC2
```

If you press CTRL/Z (either in response to the line number prompt or at the end of a program line), BASIC stops prompting and then you can enter program lines in the normal way:

```
BASIC2

SEQUENCE 100,100
100    PRINT 'GOODBYE'
200    CTRL/Z
BASIC2
```

2.3.24 SET

The SET command specifies compiler defaults. Its format is:

SET/qualifier[/qualifier...]

where:

/qualifier Can be (1) any BUILD command qualifier except for BRLRES, DSKLIB, LIBRARY, ODLRMS, RMSRES, EXTEND: int-const, or MAP, (2) any COMPILE command qualifier, or (3) the SYNTAX_CHECK qualifier, which tells BASIC to perform syntax checking after each program line is typed.

Defaults specified with the SET command remain in effect until you specify another default with the SET or LOCK command or you leave the BASIC environment, at which point the defaults return to their original settings.

For example:

```
BASIC2

SET/DOUBLE

BASIC2
```

This command specifies that all floating-point numbers are double precision. Therefore, any BUILD, COMPILE, or RUN operation uses double precision for floating-point numbers. The SHOW command displays the current default setting:

```
BASIC2

SHOW

.
.
.
Real size : DOUBLE
.
.
.

BASIC2
```

If you execute a SET command without qualifiers, the defaults return to their original settings.

2.3.25 SHOW

The SHOW command displays the current compiler defaults. Its format is:

```
SHOW
```

See Section 2.1 for an example and explanation of its output.

2.3.26 UNSAVE

The UNSAVE command deletes a file from disk. Its format is:

```
UNSAVE file-spec
```

where:

file-spec	Is the name of the file you want to delete. If you do not specify a file name, the UNSAVE command deletes the disk file associated with the program currently in memory.

For example:

```
BASIC2

OLD PROG1

BASIC2

UNSAVE

BASIC2
```

In this example, the OLD command copies a program named PROG1.B2S from disk to memory. The UNSAVE command deletes the latest version of that program from disk.

You can delete a BASIC program other than the one currently in memory by specifying its file name:

```
UNSAVE PROG2
```

This command deletes the latest version of the file named PROG2.B2S.

To delete a file other than a BASIC program, specify the file name and file type. For example:

```
UNSAVE PROG2.OBJ
```

This command deletes the latest version of the object module generated from the compilation of PROG2.B2S.

- If you answer "1", all the statements in the first CASE statement execute. The INPUT statement asks you for the number of records to skip. Your response is assigned to a variable that is used as the parameter in the MAGTAPE function.
- If you answer "2", all the statements in the second CASE statement execute. The INPUT statement asks you for the number of records to backspace. Your response is assigned to a variable that is used as the parameter in the MAGTAPE function.
- If you answer "3", all the statements in the third CASE statement execute. The INPUT statement asks you for the number of records to read. Your response is assigned to a variable that is used in a loop to retrieve and display records.
- If you choose an invalid selection number, a message is displayed and control transfers to the label Choice.
- The last MAGTAPE function compares the values of the MAGTAPE function and the Tape Status Word to check for the end of the file.

4.4.1.7 Truncating a File

You can truncate the remaining records on a tape using the MAGTAPE function to write two EOF marks. For example:

```
10      ON ERROR GOTO 19000
        !
        MAP (FUN) STRING RECORD1 = 128%,        &
                  STRING RECORD2 = 128%    &
                  STRING RECORD3 = 128%,   &
                  STRING RECORD4 = 128%
        !
        OPEN 'XX:' FOR INPUT AS FILE #4%, MAP FUN
        !
        ! Skip 27 records
        !
        I% = MAGTAPE(4%,27%,4%)
        GET #4%
        PRINT RECORD1, RECORD2, RECORD3, RECORD4
        PRINT 'Do you want to delete the remaining records '
        INPUT 'on this tape <YES/NO>';ANS$
                !
                IF ANS$ = 'NO'
                        !
                        ! Rewinds the tape to its BOT mark
                        !
                        THEN
                                I% = MAGTAPE(3%,0%,4%)
                        !
                        ! Writes an EOT mark
                        !
                        ELSE
                                I% = MAGTAPE(2%,0%,4%)
                                I% = MAGTAPE(2%,0%,4%)
                END IF
                !
                GOTO 32766
                !
```

```
19000   ! Error Handler
        !
        PRINT 'Unexpected error: ';ERT$(ERR)
        RESUME 32766
        !
32766   CLOSE #4%
32767   END
```

In this example:

- The MAP statement allocates storage for four 128-byte records.
- The OPEN statement opens a tape for input on channel 4.
- The first MAGTAPE function positions the tape the length of 27 records. The current tape position is at the beginning of the twenty-eighth record.
- The GET statement retrieves the record at the current tape position.
- The PRINT statement displays the current record.
- The INPUT statement asks you if you want to truncate the remaining records. If you do want to truncate the remaining records, the second MAGTAPE function writes an EOT mark to the current tape position. If you do not want to truncate the remaining records, the first MAGTAPE function rewinds the tape.

4.5 Device-Specific I/O to Disk

A disk is a relatively expensive medium that can provide large amounts of online storage. The records stored on a disk can be accessed randomly; therefore, you can access those records quickly. However, storing data on disk can be inefficient if each record does not fill the amount of storage allocated for it.

When you read and write data to a disk, you must think of the data as 512-byte blocks instead of individual records in a file. When you write records to disk, you can store as many records as can fit in a 512-byte disk block. The maximum number of blocks on a disk depends on the type of disk you use. You specify the location and format of the records in each block by blocking and deblocking records with MAP, MAP DYNAMIC, REMAP, or MOVE statements. See the *BASIC User's Guide* for information on these statements.

If you do not use RMS–11, the Task Builder does not need to link to RMS–11 libraries. Therefore, add the NOVIR qualifier to the BASIC BUILD command to generate Task Builder CMD and ODL files that do not access RMS–11 libraries. You can also specify no RMS–11 code by executing the BASIC RMSRES command or by adding the RMSRES qualifier to the BASIC BUILD command with NONE as the argument before you generate CMD and ODL files for your program. By doing this, you free at least 8K words of virtual address space and substantially decrease the time it takes to link your program.

To use a disk, do the following:

1. Issue the ALLOCATE command to allocate the disk drive.
2. Place the disk in the disk drive and spin it up.
3. On RSX–11M–PLUS systems, issue the MOUNT command with the FOREIGN qualifier.

See Section 4.2.1 for the format of the ALLOCATE command. Substitute the physical or logical name of the disk drive you use for ddnn:.

Consult your disk drive's operation manual for information explaining how to place your disk in the disk drive and spin it up.

See Section 4.2.2 for the format of the MOUNT command. A disk's volume label can be from 1 to 12 characters. Volume labels are mandatory for nonprivileged users.

For example:

DCL

```
ALLOCATE DK2:
MOUNT /FOREIGN DK2: PRG
```

MCR

```
ALLOCATE DK2:
MOUNT DK2:PRG/FOREIGN
```

This example allocates the disk drive DK2: and specifies a foreign file structure. The volume label is PRG.

You can use the following BASIC statements to handle file operations on disk:

- OPEN
- PUT
- GET
- CLOSE

4.5.1 Opening a Disk

You open a channel to a disk using the OPEN statement with the VIRTUAL organization clause. Use this format of the OPEN statement:

```
OPEN 'dev:' AS FILE [#]chnl-exp ORGANIZATION VIRTUAL [,clause(s)]
```

where:

dev:	Is the physical name and number or logical name of the drive on which your disk is placed. Note that you do not include a file name.
chnl-exp	Is the number of the channel on which the disk is open.
clause(s)	Can be the RECORDSIZE or MAP clauses.

After you finish file operations, you should explicitly close all open files with the CLOSE statement. See Section 4.3.7 for the format of the CLOSE statement. When BASIC executes the CLOSE statement, it releases any allocated record buffer space.

4.5.2 Writing Records

You use the PUT statement to write records to a disk. Each PUT operation writes an integral number of 512-byte blocks. You can write records either randomly by block number or sequentially by omitting the RECORD clause to the PUT statement.

The format of the PUT statement is:

```
PUT [#]chnl-exp [,RECORD int-exp]
```

where:

chnl-exp	Is the number of the channel on which the disk is open.
int-exp	Is the number of the block where you want to write the record.

If your record size is not 512 bytes, you waste space. Pad the unused space with FILL statements to make sure that no unexpected data is written to disk.

You also can write records that are longer than a block in length. For example, if your record is five blocks long (a total of 2560 bytes) and you use a RECORD 1% clause, this record is written to the first through fifth disk blocks. If you use a RECORD 2% clause to write a second record of the same size, the record is physically written to the sixth through tenth disk blocks.

```
10      ON ERROR GOTO 19000
        !
        DECLARE WORD CT
        !
        MAP (GAME) LONG REC, STRING FILL = 508%
        !
        OPEN 'DK0:' AS FILE #9%, ORGANIZATION VIRTUAL, MAP GAME
        !
        ! Use the counter index to specify RECORD value
        !
        FOR CT = 1% TO 15%
                INPUT REC
                MOVE TO #9%, REC
                PUT #9%, RECORD CT
        NEXT CT
        !
        GOTO 32766
        !
19000   ! Error Handler
        !
        PRINT 'Unexpected error: ';ERT$(ERR)
        RESUME 32766
        !
32766   CLOSE #9%
32767   END
```

In this example:

- The OPEN statement opens a channel to the disk on DK0:. There is no RECORDSIZE clause, so the record size defaults to 512 bytes.
- The FOR-NEXT loop writes 15 records to disk.
- The RECORD clause uses the value of each iteration of the loop to specify which block the record is written to. Therefore, during the first loop the PUT statement writes the first record to the first block, during the second loop the PUT statement writes the second record to the second block, and so on up to block 15.

4.5.3 Adding New Records

To add new records to a disk, simply write them to an available block using the PUT statement with the RECORD clause. You must know which disk blocks are available to prevent writing over existing records. Note that BASIC does not warn you if you write over an existing record.

If you wrote four 128-byte records (a total of 512 bytes) to each of 30 blocks, you should be able to add records to the thirty-first block:

```
PUT #4%, RECORD 31%
```

4.5.4 Reading Records

You can read records from disk using the GET statement. Each GET operation retrieves an integral number of 512-byte blocks. The format of the GET statement is:

GET [#]chnl-exp [,RECORD int-exp]

where:

chnl-exp — Is the number of the channel on which the disk is open.

int-exp — Is the number of the block where the record is stored.

If you do not specify a RECORD clause, each GET operation sequentially retrieves the next record.

If you define the size of each record to be 128 bytes, you can store four records in each disk block. One GET operation makes four records available for processing. For example:

```
10      ON ERROR GOTO 19000
        !
        MAP (VIR) STRING A = 128,       &
                  B = 128,              &
                  C = 100,              &
                  FILL = 28,            &
                  D = 128
        !
        OPEN 'DK0:' AS FILE #4%, ORGANIZATION VIRTUAL, MAP VIR
        !
        GET #4%, RECORD 6%
        PRINT A, B, C, D
        !
```

```
        GOTO 32766
        !
19000   ! Error Handler
        !
        PRINT 'Unexpected error: '$ERT$(ERR)
        RESUME 32766
        !
32766   CLOSE #4%
32767   END
```

In this example, the GET statement retrieves the first 512 bytes located in the sixth block. Note that the FILL statement in the MAP declaration pads 28 bytes of unused space in the third record.

Because each GET operation transfers an integral number of 512-byte disk blocks, your program must perform record blocking and deblocking. Use the MAP, MAP DYNAMIC, REMAP, or MOVE statements to do this.

MAP statements allocate static memory. REMAP statements can redefine the memory allocated with MAP statements. For example:

```
10      MAP (VIR) STRING TOTAL_RECORD = 512%
        !
        OPEN 'DK0:' AS FILE #1%, ORGANIZATION VIRTUAL, MAP VIR
        !
        MAP DYNAMIC (VIR) STRING CURRENT_RECORD
                        !
                        FOR I% = 0% TO 3%
                        REMAP (VIR) STRING FILL = I% * 128%, CURRENT_RECORD
                        PRINT CURRENT_RECORD
                NEXT I%
```

In this example:

- Line 10 declares the storage and data type for 512 bytes.
- The OPEN statement opens the disk on channel one and references the MAP declared in line 10.
- The MAP DYNAMIC statement declares the data type and name of CURRENT_RECORD. At each loop iteration, the position of this variable in the buffer is redefined with the REMAP statement.

This example uses MOVE statements to block and deblock records:

```
10      ON ERROR GOTO 19000
        !
        OPEN 'DK0:' AS FILE #3%, ORGANIZATION VIRTUAL, RECORDSIZE 512%
        !
        DECLARE WORD CT, STRING REC
        !
                FOR CT = 1% TO 15%
                        GET #3%, RECORD CT
                        MOVE FROM #3%, REC
                        PRINT REC
                NEXT CT
```

```
        !
        GOTO 32766
        !
19000   ! Error Handler
        !
        PRINT 'Unexpected error: ';ERT$(ERR)
        RESUME 32766
        !
32766   CLOSE #3%
32767   END
```

In this example:

- The OPEN statement opens the disk on channel three. The record size and record buffer size is 512 bytes.
- The FOR-NEXT loop retrieves one 512-byte record from blocks 1 through 15.
- The MOVE statement moves each record from the record buffer and assigns it to the string variable RECORD.
- The PRINT statement displays the value of RECORD.

4.5.5 Locating Records

There is no BASIC statement you can use to find a record on disk. To locate records, you must know the contents of each block and then read the records in the block you specify with the RECORD clause added to the GET statement.

4.5.6 Deleting Records

There is no BASIC statement you can use to delete or truncate records on disk. To delete a record, you must write over it.

4.5.7 Dismounting a Disk

After you finish using a disk, you must do the following:

1. If you use the MOUNT command, issue the DCL DISMOUNT or MCR DMO command.
2. Issue the DEALLOCATE command.
3. Spin down and remove your disk.

Substitute the physical or logical name of the disk drive you use for ddnn: in the DISMOUNT, DMO, and DEALLOCATE commands. Consult your system manager to learn how to spin down and remove your disk.

Caution

Do not remove a disk until the message "Dismount complete" displays on the console terminal. If you do not wait for this message before removing the disk, you could corrupt the data on it.

4.6 Analyzing Existing Files

BASIC provides two functions for analyzing the characteristics of existing files:

- The FSP$ function
- The FSS$ function

To use the FSP$ function, the program must open the file in question with an ORGANIZATION UNDEFINED clause. The FSS$ requires only the file name as an argument and does not need to access the file itself.

The following sections describe these two functions.

4.6.1 The FSP$ Function

The FSP$ function returns the file organization data for an opened file. This function is intended for files opened with an ORGANIZATION UNDEFINED clause, and your program must execute it immediately after the OPEN statement. The format of the FSP$ function is:

```
FSP$(channel-number)
```

For example:

```
10      MAP (A) A$ = 32
        MAP (A) A%(15)
20      OPEN "FILE.DAT" FOR INPUT AS FILE #1,   &
          ORGANIZATION UNDEFINED,               &
          ACCESS READ
30      A$ = FSP$(1)
```

In this program FSP$ generates the following values:

- A%(0) returns file characteristics. The high byte contains the RMS–11 Record attributes (RAT) field. The low byte contains the RMS–11 Organization (ORG) and Record Format (RFM) fields.
- A%(1) returns the RMS–11 maximum record size (MRS) field.
- A%(2) and A%(3) return the RMS–11 allocation quantity (ALQ) field.
- A%(4) returns the RMS–11 bucketsize (BKS) field for files on disk or the RMS–11 blocksize (BLS) field for files on tape.
- A%(5) returns the number of keys.

- A%(6) and A%(7) return the RMS–11 maximum record number (MRN) field if the file is relative.
- A%(8) and A%(9) return the current block or record number, or both.

When you open a file with an ORGANIZATION UNDEFINED, you must (1) open the file FOR INPUT only and (2) include the RMS code for all file organizations in your task. That is, use the /SEQ, /REL, and /IND qualifiers to the BUILD command to make sure your program can access the RMS routines for the file you are opening.

4.6.2 The FSS$ Function

FSS$ performs a file name scan on the argument string. Its format is:

```
FSS$ (A$, B%)
```

where:

A$ Is the file name string.

B% Is the starting position of the scan in the string.

The output is a 30-character string encoded as shown in Tables 4–4 to 4–6.

Table 4–4: File Name String: Flag Word Bytes 1–30

Byte	Meaning
1	Job number multiplied by two.
2	Version number. If the version number is undefined, the byte returns zero.
3–4	Last three characters of the file name.
5–6	UIC.
7–10	File name in Radix–50 format.
11–12	File extension name in Radix–50 format.
13–21	Undefined.
22	Protection code if byte 21 is nonzero.
23–24	Device name if specified. If the device name is logical and not translatable, byte 23 contains the first two characters in Radix–50 format; byte 24 contains the last two characters.
25	Unit number of the device. If no device is given, byte 25 returns a zero.
26	255 if the unit number was specified.
27–28	Flag word 1. See Table 4–5.
29–30	Flag word 2. See Table 4–6.

Table 4–5: File Name String: Scan Flag Word 1

Flag Word 1: where S0% = M%(27%) + SWAP%(M%(28%))

Bit	Logical Test	Meaning
0–6		Not used.
7	(S0% and 128%)<>0%	A semicolon (;) was found, implying a version number.
8	(S0% and 256%)<>0%	A file name was found in the source string (and is returned in Radix–50 format in bytes 7 through 10).
	(S0% and 256%)=0%	No file name was found.
9	(S0% and 512%)<>0%	A dot was found in the source string.
	(S0% and 512%)=0%	No dot was found in the source string, implying that no extension was specified.
10	(S0% and 1024%)<>0%	An UIC was found in the source string.
	(S0% and 1024%)=0%	No UIC was found.
11	(S0% and 2048%)<>0%	A left angle bracket (<) was found in the source string, implying that a protection code was found.
	(S0% and 2048%)=0%	No left angle bracket (<) was found, implying that no protection code was specified.
12	(S0% and 4096%)<>0%	A colon (but not necessarily a device name) was found.
	(S0% and 4096%)=0%	No colon was found, implying that no device was specified.
13	(S0% and 8192%)<>0%	A logical name was specified.
	(S0% and 8192%)=0%	A device name was not specified.
15	S0%<0%	The source string contained wild card characters in the file name, extension, or UIC field. In addition, the specified device name does not correspond to any logical device assignments. The program must test bits of flag word 2 for wild card characters and device name.

Table 4–6: File Name String: Scan Flag Word 2

Flag Word 2: where S1% = M%(29%) + SWAP%(M%(30%))

Bit	Logical Test	Meaning
0	(S1% and 1%)<>0%	File name was found in the source string.
	(S1% and 1%)=0%	No file name was found, and bits 1 and 2 return zero.
1	(S1% and 2%)<>0%	File name was an asterisk (*) and is returned in bytes 7 through 10 as the Radix–50 representation of the string "??????".
	(S1% and 2%)=0%	File name was not an asterisk.

(continued on next page)

Table 4–6 (Cont.): File Name String: Scan Flag Word 2

Flag Word 2: where S1% = M%(29%) + SWAP%(M%(30%))

Bit	Logical Test	Meaning
2	(S1% and 4%)<>0%	File name contained at least one question mark (?).
	(S1% and 4%)=0%	File name did not contain any question marks.
3	(S1% and 8%)<>0%	A dot (.) was found.
	(S1% and 8%)=0%	No dot was found, implying that no extension was specified, and bits 4, 5, and 6 return zero.
4	(S1% and 16%)<>0%	An extension was found (the field after the dot was not null).
	(S1% and 16%)=0%	No extension was found (the field after the dot was null), and bits 5 and 6 return zero.
5	(S1% and 32%)<>0%	The extension was an asterisk (*) and is returned in bytes 11 and 12 as the Radix–50 representation of the string "???".
	(S1% and 32%)=0%	The extension was not an asterisk.
6	(S1% and 64%)<>0%	The extension contained at least one question mark (?).
	(S1% and 64%)=0%	The extension did not contain any question marks.
7	(S1% and 128%)<>0%	A UIC was found.
	(S1% and 128%)=0%	No UIC was found, and bits 8 and 9 return zero.
8	(S1% and 256%)<>0%	UIC was in the form of [*,PROG] and is returned in byte 6 as 255.
	(S1% and 256%)=0%	UIC was not an asterisk.
9	(S1% and 512%)<>0%	UIC was in the form [PROJ,*] and is returned in byte 5 as 255.
	(S1% and 512%)=0%	UIC was not an asterisk.
10	(S1% and 1024%)<>0%	A protection code was found.
	(S1% and 1024%)=0%	No protection code was found.
11	(S1% and 2048%)<>0%	The protection code set as default by the current job was used.
	(S1% and 2048%)=0%	The protection code is the default (<60>) or that found in the source string.
12	(S1% and 4096%)<>0%	A colon (but not necessarily a device name) was found.
	(S1% and 4096%)=0%	No colon or device was found, and bits 13, 14 and 15 return zero.

(continued on next page)

Table 4–6 (Cont.): File Name String: Scan Flag Word 2

Flag Word 2: where S1% = M%(29%) + SWAP%(M%(30%))

Bit	Logical Test	Meaning
13	(S1% and 8192%)<>0%	A device name was found.
	(S1% and 8192%)=0%	No device name was found; bits 14 and 15 return zero.
14	(S1% and 16384%)<>0%	A logical device name was specified.
	(S1% and 16384%)=0%	An actual device name was specified; bit 15 returns zero.
15	S1%<0%	The specified device name is a logical name and is not assigned to an actual device. The logical name is returned in bytes 23 through 26 as a Radix–50 string.
	S1%=0%	A physical device is assigned to the physical or logical device name. This physical device name is returned in bytes 23 and 24; unit information is returned in bytes 25 and 26.

For example:

```
10        DIM Y%(30%)
20        LINPUT 'ENTER FILE NAME STRING';A$
          INPUT 'ENTER OFFSET';B%
30        Y$ = FSS$ (A$, B%)
          CHANGE Y$ TO Y%
40        PRINT J%, Y%(J%) FOR J% = 1% TO 30%
32767     END

RUN

ENTER FILE NAME STRING? FILE.EXT
ENTER OFFSET? 1
1         32
2         0
3         0
4         0
5         1
6         1
7         244
8         38
9         64
10        31
11        20
12        35
13        0
14        0
15        0
```

```
16      0
17      0
18      0
19      0
20      0
21      0
22      0
23      68
24      66
25      0
26      255
27      0
28      23
29      153
30      48
```

In this program output:

- Byte 1 (32) is the job number multiplied by 2.
- Bytes 5 and 6 are the UIC (1,1).
- Bytes 7 through 12 are the file name and extension in Radix–50 format.
- Bytes 23 and 24 are the device name in Radix–50 format.
- Byte 26 indicates that the unit number was specified.
- Byte 28 (23) is the binary number 00010111. The bits indicate the following:
  - The file name was found (bit 8)
  - A dot was found (bit 9)
  - The UIC was found (bit 10)
  - A colon was found (bit 12)
- Byte 29 (153) is the binary number 010011001. The bits indicate the following:
  - The file name was found (bit zero)
  - A dot was found (bit 3)
  - The extension was found (bit 4)
  - The UIC was found (bit 7)
- Byte 30 (48) is the binary number 110000. The bits indicate that:
  - A colon was found (bit 12)
  - The device name was found (bit 13)

Chapter 6
Debugging BASIC Programs

Debugging is the process of eliminating the errors in logic in your programs. BASIC provides you with the BASIC debugger to help you debug your programs.

The debugger has a set of commands that allow you to specify breakpoints. Breakpoints temporarily halt the execution of your program, at which point you can do the following:

- Look at program errors
- Display values describing storage allocation
- Examine the contents of program variables
- Assign new values to program variables
- Display values describing file status characteristics

6.1 The Debugger

You can access the debugger by adding the DEBUG qualifier to the BASIC RUN command or by following these steps:

1. Compile the program module you want to debug, adding the DEBUG qualifier to the COMPILE command.
2. Create CMD and ODL files for the Task Builder with the BASIC BUILD command.
3. Link your program at the RSX–11M/M–PLUS system command level by using either the DCL LINK/BASIC command or by invoking the Task Builder with the MCR TKB acronym.
4. Execute your program at the RSX–11M/M–PLUS system command level using the RUN command.

When you use the DEBUG qualifier, the Task Builder links the debugger program module from the BASIC Object Time System to your program. This increases the size of your task by about 4K words.

Follow the previous steps to debug any program that uses subprograms. However, you can make the object modules for subprograms available for debugging if you enter the SET/DEBUG command before using the LOAD command. Then, use the RUN/DEBUG command to execute the main program that calls the subprograms. Use the DEBUG qualifier to the BASIC RUN command to debug any program in the BASIC environment that is small enought to fit in memory without being overlaid.

When you execute your program, execution stops at the first line in the first program module compiled with the DEBUG qualifier. For example, if your main program calls three subprograms but you compile only the three subprograms with the DEBUG qualifier, execution stops at the first subprogram called.

Once BASIC stops execution, the debugger displays its prompt, at which you can enter debugger commands:

```
DEBUG:filename
#
```

where:

filename	Is the name of the currently executing program module that has been compiled with the DEBUG qualifier.
#	Is the debugger prompt. You can type debugger commands in response to the debugger prompt. Debugger commands are listed in Table 6–1.

Type the CONTINUE command after you enter a debugger command to pass control to your program, resume program execution, and execute the debugger command you enter. For example:

```
DEBUG:MYPROG
#BREAK 10
#CONTINUE
```

In this example:

- The debugger displays the file name of the currently executing program module MYPROG.
- The # prompt indicates control has passed from your program to the debugger. The BREAK 10 command tells the debugger to set a breakpoint at line 10 of MYPROG.
- The CONTINUE command passes control to your program, resumes program execution, and executes the BREAK command.

Table 6–1 lists the commands you can use in response to the debugger prompt.

Table 6–1: Debugger Commands

Command	Function
BREAK	Stops program execution at the first executable statement in the currently executing program module. Breakpoints set by any form of the BREAK command remain in effect until you disable the breakpoint with the UNBREAK command.
BREAK lin-num	Stops program execution at the line number in the currently executing program module. You can set multiple breakpoints by separating multiple line numbers with commas. You can specify a maximum of 10 breakpoints. Program execution stops and control passes to the debugger when BASIC executes the line number you specify.
BREAK lin-num.stat-num	Stops program execution at the line number and the number of the statement in the currently executing program module.
BREAK lin-num.stat-num;mod-nam	Stops program execution at the line number and the number of the statement in the module you specify.
BREAK ON {CALL,DEF,LOOP}	Stops on CALL statements, user-defined functions, or loops in the currently executing program module.
CONTINUE	Passes control to your program, resumes program execution, and executes BREAK and TRACE commands, if entered.
CORE	Displays the number of words in memory that is currently allocated for your task.
ERL	Displays the line number of the module that was executing when the last error was found.
ERN	Displays the name of the module that was executing when the last error was found.
ERR	Displays the number of the last error.
EXIT	Returns control to the RSX–11M/M–PLUS system command level.
FREE	Displays the number of words in memory that is currently available for free space.
I/O BUFFER	Displays the number of words in memory that is currently allocated for I/O buffer space.
LET	Changes the contents of a variable. You cannot create new variables.
PRINT	Displays the contents of a variable.
RECOUNT	Displays how many characters were transferred by the last input operation.
REDIRECT term-nam	Sends all I/O during a debugging session to a specified terminal.
STATUS	Depending on the error, displays a value representing one of the following: – The RMS–11 primary status field (STS). See the *RMS–11 MACRO User's Guide* for more information. – The device characteristics after an RMS–11 OPEN file operation set by the DEV field of the FAB. See the *RMS–11 MACRO User's Guide* for more information.

(continued on next page)

Table 6–1 (Cont.): Debugger Commands

<table>
<tr><th>Command</th><th>Function</th></tr>
<tr><td>STATUS (Cont.)</td><td>
– In the event of a directive error, the Directive Status Word ($DSW) and its corresponding error code. See the RSX–11M Mini Reference or the RSX–11M–PLUS Mini Reference for the error codes.
– The STATUS field of a QIO. See the RSX–11M/M–PLUS I/O Drivers Reference Manual for more information.
– The first word of either the GETLUN or GLUN_$ directive describing device characteristics. See the RSX–11M/M–PLUS Executive Reference Manual.
– For an OPEN file operation with no errors, BASIC sets the value of the status word:
<table>
<tr><th>Status Value</th><th>Bit Set</th><th>Meaning</th></tr>
<tr><td>1</td><td>0</td><td>Record-oriented device</td></tr>
<tr><td>2</td><td>1</td><td>Carriage-control device</td></tr>
<tr><td>4</td><td>2</td><td>Terminal device</td></tr>
<tr><td>8</td><td>3</td><td>Directory device</td></tr>
<tr><td>16</td><td>4</td><td>Single directory device</td></tr>
<tr><td>32</td><td>5</td><td>Sequential device</td></tr>
<tr><td>64</td><td>6</td><td>Mass storage device</td></tr>
<tr><td>128</td><td>7</td><td>User-mode diagnostics supported</td></tr>
<tr><td>256</td><td>8</td><td>Massbus device</td></tr>
<tr><td>512</td><td>9</td><td>Unit software write-locked</td></tr>
<tr><td>1024</td><td>10</td><td>Input spooled device</td></tr>
<tr><td>2048</td><td>11</td><td>Output spooled device</td></tr>
<tr><td>4096</td><td>12</td><td>Pseudo device</td></tr>
<tr><td>8192</td><td>13</td><td>Device mountable as a communication</td></tr>
<tr><td>16384</td><td>14</td><td>Device mountable as a Files–11 device</td></tr>
<tr><td>–32768</td><td>15</td><td>Device mountable</td></tr>
</table>
</td></tr>
<tr><td>STEP</td><td>Executes the next statement in the currently executing program module. If you enter a carriage return at the debugger prompt (#), it acts like a STEP command.</td></tr>
<tr><td>STEP number</td><td>Executes the number of statements you specify before stopping program execution and passing control to the debugger.</td></tr>
<tr><td>STRING</td><td>Displays the number of words in memory that is currently allocated for dynamic string space. This value represents dynamic memory allocation, not static memory allocation.</td></tr>
<tr><td>TRACE</td><td>Displays the line number of each program line as it executes.</td></tr>
<tr><td>UNBREAK</td><td>Disables breakpoints set by any form of the BREAK command.</td></tr>
<tr><td>UNBREAK ON</td><td>Disables breakpoints set by the BREAK ON CALL, BREAK ON DEF, and BREAK ON LOOP commands.</td></tr>
<tr><td>UNTRACE</td><td>Disables the TRACE command.</td></tr>
</table>

See Part III of the *BASIC Reference Manual* for more information on each of these debugger commands.

If you compile the program you want to debug with the LIST qualifier, BASIC generates a listing file for your program that numbers each program line and each statement in a program line. These numbers are especially useful when setting breakpoints with the BREAK debugger command.

The debugger has a few restrictions. It cannot reference external variables declared in MACRO subprograms, external constants declared in MACRO subprograms, or compile-time constants.

6.1.1 Using the Debugger in the BASIC Environment

The following section presents a sample debugging session accessing the debugger from the BASIC environment.

```
BASIC2

OLD BAL

BASIC2

LISTNH
100       PRINT 'This program keeps track of your checking balance.'
200       INPUT 'Starting balance'; BALANCE
300       INPUT 'How many checks'; NUM.CHECKS%
400       FOR I% = 1% TO NUM.CHECKS%
500               INPUT 'Amount of check'; CHECK.AMOUNT
600               BALANCE = BALANCE - CHECK.AMOUNT
700       NEXT I%
800       PRINT "Your balance is "; BALANCE
32767     END
```

This program executes with no error messages. However, no matter how many checks you write, the balance always remains at the starting value you type in. To find out where the problem lies, execute the program with the RUN/DEBUG command:

```
BASIC2

RUN/DEBUG

DEBUG:BAL

*BREAK 600

*CONTINUE

This program keeps track of your checking balance.
Starting balance? 500
How many checks? 1
Amount of check? 12.50
BREAK at line 600 statement 1

*STEP

STEP at line 600 statement 1

*PRINT BALANCE

*500
```

(continued on next page)

```
*PRINT CHECK.AMOUNT
*12.5
*CONTINUE
Your balance is 500
```

Control transfers to the debugger after BASIC executes line 600; then you can examine the values of the program variables BALANCE and CHECK.AMOUNT. When the execution of the program resumes, the program performs the subtraction using these variables, and sets the BALANCE variable to the correct value. However, the program still does not work correctly. Take a closer look at the program line performing the calculation:

```
600       BALANCE = BALANCE - CHECK.AMOUT
```

Notice that the variable CHECK.AMOUNT is misspelled in line 600. This means that BASIC subtracted a variable named CHECK.AMOUT from the variable BALANCE. Because BASIC variables are initialized to zero, line 600 did not affect the balance at all. To correct the program, re-enter line 600:

```
600       BALANCE = BALANCE - CHECK.AMOUNT
```

If you specify the compiler default SET /TYPE:EXPLICIT that requires you to explicitly declare the data types for all variables, you solve the problem of misspelling variable names.

It may be more convenient to use line numbers in your programs while debugging it, although you can specify the number of the statement in a single line number. Once you debug your program, you can eliminate unnecessary line numbers:

```
100       DECLARE INTEGER I, NUM.CHECKS, DOUBLE CHECK.AMOUNT, BALANCE
          !
          PRINT 'This program keeps track of your checking balance.'
          INPUT 'Starting balance'; BALANCE
          INPUT 'How many checks'; NUM.CHECKS
          FOR I = 1% TO NUM.CHECKS
                INPUT 'Amount of check'; CHECK.AMOUNT
                BALANCE = BALANCE - CHECK.AMOUNT
          NEXT I
          PRINT "Your balance is "; BALANCE
          END
```

6.1.2 Using the Debugger at the RSX–11M/M–PLUS System Command Level

This section presents a sample debugging session using the debugger on an RSX–11M–PLUS operating system.

The four programs used in the following example of a debugging session are:

- MAPS.B2S
- CREATE.B2S
- UPDATE.B2S
- ACTSUB.B2S

MAPS.B2S declares the data types and storage for program variables. CREATE.B2S, UPDATE.B2S, and ACTSUB.B2S use MAPS.B2S by using the %INCLUDE compiler directive. When you compile CREATE.B2S and UPDATE.B2S, MAPS.B2S is automatically included.

```
!       MAPS.B2S
!
! Declaring the data type and storage
! for the variables used in ACCT.RMS
!
MAP (ACCT) STRING ACCT.NAME = 50,       &
        ACCT.ACCOUNT = 6,       &
        SINGLE ACCT.BALANCE
!
! Declaring the data type and storage
! for the variables used in LEDGER.RMS
!
MAP(LEDGER) STRING LEDGER.NAME = 22,    &
        LEDGER.DATE = 6,        &
        LEDGER.ACCOUNT = 6,     &
        SINGLE LEDGER.AMOUNT
```

ACTSUB.B2S is a FUNCTION subprogram that opens two files named ACCT.RMS and LEDGER.RMS.

```
1       FUNCTION BYTE ACTSUB (STRING FILES.NAME, INTEGER CHANNEL_NUMBER)
        !
        !                   ACTSUB.B2S
        !
        ! This is the FUNCTION subprogram used to initialize the files.
        ! There are two parameters passed to the subprogram:
        !
        ! FILES.NAME    This is the 1- to 6-character name of the file.
        !               The file name variable returns to the main
        !               program unchanged.
        !
        ! CHANNEL_NUMBER This is the channel number of the file to be
        !                opened.  The channel number can be from 1 to
        !                12.  The file channel variable returns to the
        !                main program unchanged.
        !
        ! The value returned to the programs that call this program
        ! is a truth flag that indicates the success or failure of a
        ! file open operation. The function returns a zero if the file
        ! open operation is successful, or a -1 if it is not.
        !
        ! Use the maps declared in MAPS.B2S.
        !
        %INCLUDE 'MAPS.B2S'
        !
        ! Truth flag:
```

(continued on next page)

```
        ACTSUB = 0%
        !
2       IF FILES.NAME = 'ACCT'
                THEN
                        OPEN 'ACCT.RMS' AS FILE #CHANNEL_NUMBER, &
                                INDEXED FIXED,  &
                                ACCESS MODIFY,  &
                                ALLOW MODIFY,   &
                                MAP ACCT,       &
                                PRIMARY ACCT.NAME,      &
                                ALTERNATE ACCT.ACCOUNT  DUPLICATES CHANGES
                ELSE
                        IF FILES.NAME = 'LEDGER'
                          THEN
                                OPEN 'LEDGER.RMS' AS FILE #CHANNEL_NUMBER, &
                                    INDEXED FIXED,      &
                                    ACCESS MODIFY,      &
                                    ALLOW MODIFY,       &
                                    MAP LEDGER, &
                                    PRIMARY LEDGER.ACCOUNT DUPLICATES,  &
                                    ALTERNATE LEDGER.DATE DUPLICATES CHANGES
                                ELSE
                                ! File open not successful
                                ACTSUB = -1%
                                PRINT FILES.NAME;' Undefined in ACTSUB'
                        END IF
        END IF
        END FUNCTION
```

CREATE.B2S calls ACTSUB.B2S to open the accounting ledger files, and then asks the user for information about the account, writes the information to the open files, and finally closes the files.

```
1       !       CREATE.B2S
        !
        ! Use the variables declared in MAPS.B2S
        !
        %INCLUDE 'MAPS.B2S'
        !
        ! Declare the FUNCTION subprogram
        !
        EXTERNAL BYTE FUNCTION ACTSUB (STRING,INTEGER)
        !
        DECLARE INTEGER FILE.STATUS !FILE.STATUS is the truth flag
Open_files:     !
        ! Open the files
        !
        ! Call ACTSUB to open ACCT.RMS
        !
        FILE.STATUS = ACTSUB ('ACCT',1%)
        !
        ! Call ACTSUB to open LEDGER.RMS
        !
        FILE.STATUS = ACTSUB ('LEDGER',2%)
        !
Ask.Acct: !
          ! Enter account records
          !
          LINPUT 'Account name';ACCT.NAME
          !
          ! If the user enters a null string,
          ! close the files and exit the program
```

(continued on next page)

```
            GOTO Ask.Ledger IF ACCT.NAME = ''
            !
            LINPUT 'Account number';ACCT.ACCOUNT
            INPUT 'Account balance';ACCT.BALANCE
            !
            PUT #1 ! Write these three records to ACCT.RMS
            !
            GOTO Ask.Acct ! Loop to enter more records
            !
Ask.ledger: !
              ! Enter ledger records
              !
              LINPUT 'Ledger name';LEDGER.NAME
              !
              ! If the user enters a null string,
              ! close the files and exit the program
              !
              GOTO Done IF LEDGER.NAME = ''
              !
              LINPUT 'Ledger account';LEDGER.ACCOUNT
              LINPUT 'Ledger date';LEDGER.DATE
              INPUT 'Ledger amount';LEDGER.AMOUNT
              !
              PUT #2% ! Write these records to LEDGER.RMS
              !
              GOTO Ask.Ledger ! Loop to enter more records
              !
Done:   CLOSE #1,#2
        END
```

UPDATE.B2S calls ACTSUB.B2S and allows the user to update records in the account and ledger files. This program handles three different error conditions:

- If the record is locked, the program waits for one second and tries to retrieve the record again.
- If the record cannot be found, a message is displayed.
- If any other errors occur, the name of the program, the line number where the error occurred, and the error message for that error are displayed.

```
1       !       UPDATE.B2S
        !
        ! Use the variables declared in MAPS.B2S
        !
        %INCLUDE 'MAPS.B2S'
        !
        ! Declare the FUNCTION subprogram
        !
        EXTERNAL BYTE FUNCTION ACTSUB (STRING,INTEGER)
        !
        ! Declare the data type and storage for
        ! this program's variables
        !
        DECLARE INTEGER FILE.STATUS, CHANGE.ACCT.RECORD
        !
        MAP (FOO) STRING OLD.ACCOUNT= 6, &
                  NEW.ACCOUNT= 6, &
                  ANSWER = 1
```

(continued on next page)

```
!
! In the event of an error, transfer control to line 4
!
ON ERROR GOTO 4
!
! If the open operation was not successful,
! close ACCT.RMS
!
IF ACTSUB ('ACCT',1) = -1%
        THEN
            GOTO 5
END IF
!
! If the open operation was not successful,
! close LEDGER.RMS
!
IF ACTSUB ('LEDGER',2) = -1%
        THEN
                GOTO 5
!
2       !
! Ask for the old account numbers
!
INPUT 'Account number to change';OLD.ACCOUNT
!
! If the user enters a null string,
! close the files and exit the program
!
IF OLD.ACCOUNT = ''
        THEN
            GOTO 5
END IF
!
! Enter new account number
!
INPUT 'New account number';NEW.ACCOUNT
!
! Change this record in LEDGER.RMS too?
!
INPUT 'Change ledger record too';ANSWER
CHANGE.ACCT.RECORD = (ANSWER = 'Y')
!
! Retrieve the ACCT.RMS record
! Signal an error if the record is not found
!
3       GET #1, KEY #1 EQ OLD.ACCOUNT
!
! If ANSWER is Y, then change the
! account number record in both files
!
GOTO Get2 IF NOT CHANGE.ACCT.RECORD
!
! Change the old account number to
! the new account number
!
ACCT.ACCOUNT = NEW.ACCOUNT
!
! Change the old record in ACCT.RMS
!
! UPDATE #1
!
! Retrieve the LEDGER.RMS record
!
```

(continued on next page)

```
 Get2:  GET #2, KEY #0 EQ ACCT.ACCOUNT
        !
        ! Change the old account number to
        ! the new account number
        !
        LEDGER.ACCOUNT = NEW.ACCOUNT
        !
        DELETE #2 ! Delete the old record
        !
        PUT #2 ! Write the new record
        !
        GOTO 2 ! Loop to change more records
        !
4 Errs: SELECT ERR
        CASE 19,154
                !
                ! If the record is interlocked, then count the number
                ! of times the error occurs.  Every fifth time the error
                ! occurs, print a message, wait for one second,
                ! then try to perform the record operation again.
                !
                INTERLOCK.COUNT = INTERLOCK.COUNT +1
        PRINT 'Record is locked -- will try again'
                IF INTERLOCK.COUNT = INTERLOCK.COUNT / 5% * 5%
                        THEN SLEEP 1%
                        RESUME
                END IF
                !
        CASE 155
                ! If the record you specify does not exist,
                ! print the message:
                !
        PRINT 'That account number is not in the file'
        RESUME
                !
        CASE ELSE
                ! If any other error occurs, print the error text
                ! associated with the error code, the line
                ! number where the last error occurred, and the
                ! name of the module where the last error occurred.
                !
                PRINT ERT$(ERR)|' at line '|ERL|' in '|ERN$
                RESUME 5
        END SELECT
        !
5       !
        ! Close all open files.
        !
        CLOSE #1,#2
        END
```

To use the debugger on CREATE.B2S, compile, link, and run the program by typing the following sequence of commands:

```
BASIC/BP2

PDP-11 BASIC-PLUS-2 V2.3-00

BASIC2

OLD CREATE
```

(continued on next page)

```
BASIC2

COMPILE/DEBUG

BASIC2

OLD ACTSUB

BASIC2

COMPILE/DEBUG

BUILD/INDEXED CREATE,ACTSUB

BASIC2

EXIT
LINK/BASIC CREATE
```

The debugging session for CREATE.B2S follows:

```
RUN CREATE
```

The debugger begins the debugging session by displaying the name of the program module it is currently debugging and its prompt (#):

```
DEBUG:CREATE

*
```

The BREAK command sets a breakpoint that tells the debugger to stop execution at line 1, statement 2 in the FUNCTION subprogram named ACTSUB.B2S. The CONTINUE command tells the debugger to resume program execution.

```
*BREAK 1.2;ACTSUB

*CONTINUE
```

When BASIC reaches line 1, statement 2 in the subprogram, control passes to the debugger, and the debugger displays a message telling you where execution has stopped and then displays another prompt:

```
BREAK at line 1 statement 2 in SUB:ACTSUB
*
```

At this point in the program, the files ACCT.RMS and LEDGER.RMS have not been opened. The CORE command displays the number of words in memory that is currently allocated for your task:

```
*CORE
CORE =  15359
```

The I/O BUFFER command tells you how much storage is allocated by the I/O buffer. A value of zero indicates that no files have been opened:

```
*I/O BUFFER
I/O BUFFERS =  0
```

The STRING command tells you how many words are allocated to dynamic string storage and not static storage:

```
*STRING
STRING SPACE =  106
```

The FREE command tells you how many words are available for I/O and dynamic string operations. Generally speaking, if your program performs a lot of I/O operations and the value of the FREE command is less than 20 words, you should pre-extend the size of your task by specifying a value with the EXTTSK option in the Task Builder command file. See the *RSX–11M/M–PLUS Task Builder Manual* for more information on the EXTTSK option.

```
*FREE
FREE SPACE =  987
```

The UNBREAK command disables the breakpoint at line 1, statement 2 in ACTSUB.B2S:

```
*UNBREAK
```

The BREAK command sets another breakpoint after the files in the subprogram are opened:

```
*BREAK 2.8;ACTSUB
*CONTINUE
```

After BASIC executes statements 2 through 7, control passes to the debugger. The debugger displays a message telling you where execution has stopped and another prompt:

```
BREAK at line 2 statement 8 in SUB:ACTSUB
     2
*
```

Enter the CORE, I/O BUFFER, STRING, and FREE commands again to see how the memory allocation for your program changes:

```
*CORE
CORE =  16191

*I/O BUFFER
I/O BUFFERS =  1786

*STRING
STRING SPACE =  110

* FREE
FREE SPACE =  29
```

To make sure the files are open, use the PRINT command to return the value of ACTSUB. A value of 0 indicates that the files are open:

```
*PRINT ACTSUB

*0
```

The STEP command executes one statement at a time:

```
*STEP

STEP at line 2 statement 8 in SUB:ACTSUB
```

The CONTINUE command resumes program execution:

```
*CONTINUE

Account name? Reilly
Account number? 123456
Account balance? 560.00
Account name? RET
```

At this point in the program, these records should be written to ACCT.RMS.

```
Ledger name? Reilly
Ledger account? 123456
Ledger date? 6-30-85
Ledger amount? 45.60
Ledger name? RET
```

At this point in the program, these records should be written to LEDGER.RMS. CREATE.B2S appears to be working as expected.

To use the debugger on UPDATE.B2S, type the following sequence of commands:

```
BASIC/BP2

PDP-11 BASIC-PLUS-2 V2.3-00

BASIC2

OLD UPDATE

BASIC2

COMPILE/DEBUG

BASIC2

OLD ACTSUB

BASIC2

COMPILE/DEBUG

BASIC2
```

(continued on next page)

```
BUILD/INDEXED UPDATE,ACTSUB

BASIC2

EXIT
LINK/BASIC UPDATE
```

The debugging session for UPDATE.B2S follows:

```
RUN UPDATE
```

The debugger begins the debugging session by displaying the name of the program module it is currently debugging and its prompt (#):

```
DEBUG: UPDATE

*
```

The BREAK command sets a breakpoint at line 2, statement 1 in UPDATE.B2S:

```
*BREAK 2.1;UPDATE

*CONTINUE
```

When BASIC reaches line 2, statement 1 in UPDATE.B2S, control passes to the debugger:

```
BREAK at line 2 statement 1
*
```

The TRACE command displays the line number of each program line as it executes:

```
*TRACE

*CONTINUE

Account number to change? 12345
at line 2 statement 2

at line 2 statement 5

New account number? 234567
at line 2 statement 6

Change ledger record too?
```

It is not clear whether you should answer the previous question with N, Y, NO, or YES. Change the question in the program to supply the acceptable choices. For example:

```
INPUT 'Change ledger record too <Y/N>';ANSWER
```

The program continues:

```
Change ledger record too <Y/N>? Y

at line 2 statement 7

at line 3 statement 1

at line 4 statement 1

at line 4 statement 1

at line 4 statement 2
at line 4 statement 9

at line 4 statement 10

That account number is not in the file

at line 4 statement 11

at line 3 statement 1

at line 4 statement 1

at line 4 statement 1

CTRL/C
DCL>ABORT
```

Note that the account number to change is incorrect. An infinite loop was created when the RESUME statement in line 4, statement 11 transferred control to the program line (line 3) that caused the error. A CTRL/C was pressed to abort the debugging session.

This debugging session tells you the following:

- The files in the subprogram were opened and the records were written to them.
- Line 2, statement 6 should be rewritten to be self-documenting.
- Line 4, statement 11 should be rewritten to:

  RESUME 2

6.1.3 Debugger Error Messages

This section lists PDP–11 BASIC–PLUS–2 debugger error messages and their possible causes. See the *BASIC Reference Manual* for information on debugger commands.

?What?

Explanation: The debugger does not understand the command.

User Action: Check the spelling, syntax, and validity of the command.

%Stop at line N in subprogram X

Explanation: A STOP statement in the program halted program execution at line <n> in the subprogram named <x>.

User Action: Type CONTINUE to resume execution.

%Illegal syntax in LET

Explanation: The format of the LET debugger command is incorrect; you tried to create a new variable or assigned an expression to the variable.

User Action: Change the LET command. You cannot create a new variable with the LET command. You can specify a constant or variable as the new value but not as an expression. The LET command allows you to test or change only one variable at a time. To change or test multiple variables, use multiple LET commands.

%Illegal syntax in PRINT

Explanation: The format of the PRINT debugger command is incorrect, or you tried to print a variable not in the currently executing module.

User Action: Change the PRINT command. You cannot print a variable that does not exist in the currently executing program module. The PRINT command allows you to display only one variable at a time. To display multiple variables, use multiple PRINT commands.

%On error entry in debugger

Explanation: A CTRL/C trap or program error has stopped program execution and the debugger.

User Action: None.

%Can't CONTINUE or STEP

Explanation: The program encountered an error it could not handle and execution stopped.

User Action: Type the EXIT command to leave the debugger. Program execution cannot resume.

%Data error in LET or PRINT

Explanation: The debugger encountered a program conversion error while processing a LET or PRINT command.

User Action: Change the LET or PRINT command format.

%Bad line spec in (UN)BREAK

Explanation: You specified a nonexistent line, used incorrect syntax in specifying a program or subprogram, or used incorrect syntax when listing multiple breakpoints.

User Action: Check the syntax and change the BREAK or UNBREAK command.

%No room

Explanation: You specified too many breakpoints for a BREAK or UNBREAK command.

User Action: None. The debugger accepts as many as 10 breakpoints and ignores the rest.

%Cannot open device

Explanation: You specified a device for the REDIRECT command that does not exist or cannot be used.

User Action: Change the device specification to one that is available.

Appendix A
Compile-Time Error Messages

This appendix describes compile-time and compiler command errors, their causes, and the user action required to correct them.

A.1 Compile-Time Errors

In addition to diagnosing compile-time errors, BASIC does the following:

- Indicates the program line that generated the error or errors
- Displays the program line that generated the error or errors
- Shows you the location of the error or errors and assigns a number to each location for future reference
- Displays the statement number within the line, the location number as previously displayed, and the error message text. This is repeated for each error in the line.

BASIC repeats this procedure for each error diagnosed during compilation. The error message format for compile-time errors is:

```
? S <n>, <n>: <message>
```

where:

S <n>	Is the nth statement within the displayed line (the statement containing the error).
<n>:	Is the nth error within the line's "picture".
<message>	Is the text of the error message.

For example:

```
Error on line 10

        10 DECLARE REAL BYTE A, A
.........................1.......2

? S 1, 1:   conflicting data type specifications
? S 1, 2:   illegal multiple definition of name A
```

This display tells you that two errors were detected on line 10; BASIC displays the line containing the error, then prints a "picture" showing you where the errors were detected. In the example, the picture shows a "1" under the keyword BYTE and a "2" under the second occurrence of variable A. The following line shows you:

Example	Meaning
S 1	Which statement in the line contained the error
1	Which error in the line's "picture" is referred to by the mnemonic
conflicting...	The message associated with that error

In the example above, the first error message tells you that there are two contradictory data type keywords in the statement. The next line shows you the same type of information for the second error; in this case, the compiler detected multiple declarations of variable A.

If a compilation causes an error of type INFORMATIONAL or WARNING, the compilation continues and produces an object module. If a compilation causes an error of severity ERROR, the compilation continues but produces no object module. If a compilation causes an error of severity FATAL, the compilation aborts immediately.

The following is an alphabetical list of BASIC compile-time error messages:

actual argument must be specified

Explanation: ERROR — A DEF function reference contains a null argument, for example, FNA(1,,2).

User Action: Specify all arguments when referencing a DEF function.

an internal coding error has been detected. Submit an SPR

Explanation: ERROR — An internal error occurred during the compilation.

User Action: Submit an SPR and include all relevant information.

APPEND/SCRATCH access inconsistent with file organization

Explanation: ERROR — An OPEN statement specifies ACCESS APPEND or SCRATCH for a relative or indexed file.

User Action: Change the ACCESS clause; ACCESS APPEND or SCRATCH is valid only for sequential files.

array <name> not allowed in DEF declaration

Explanation: ERROR—The parameter list for a DEF function definition contained an entire array.

User Action: Remove the array specification; you cannot pass an entire array as a parameter to a DEF function.

attempt to sequence over existing statement

Explanation: WARNING — A SEQUENCE command specifies a starting line number that already exists in the BASIC source program in memory.

User Action: Specify a starting line number higher than any existing line or delete the old statement before using the SEQUENCE command.

attributes of overlaid variable <name> don't match

Explanation: WARNING—A variable name appears in more than one overlaid MAP; however, the attributes specified for the variable are inconsistent.

User Action: If the same variable name appears in multiple overlaid MAPs, the attributes (for example, data type) must be identical.

attributes of prior reference to <name> don't match

Explanation: WARNING — A variable or array is referenced before the MAP that declares it. The attributes of the referenced variable do not match those of the declaration.

User Action: Make sure that the variable or array has the same attributes in both the reference and the declaration.

bad line number pair

Explanation: ERROR—A compiler command specifies nonexistent line numbers or a pair of line numbers out of sequence, for example, LIST 100–2.

User Action: Specify only existing line numbers and specify the numbers in correct numeric order when specifying a range.

bound cannot be specified for array

Explanation: ERROR — An EXTERNAL statement declaring a SUB or FUNCTION subprogram specifies bounds in an array parameter. For example:

```
EXTERNAL SUB XYZ (LONG DIM(1,2,3))
```

User Action: Remove the array parameter's bound specifications. When declaring an external subprogram you can specify only the number of dimensions for an array parameter. For example:

```
EXTERNAL SUB XYZ (LONG DIM(,,))
```

bounds must be specified for array

Explanation: ERROR — The program contains an array declaration that does not specify the bounds (maximum subscript value). For example:

```
DECLARE LONG A(,)
```

User Action: Supply bounds for the declared array. For example:

```
DECLARE LONG A(50,50)
```

built in function not supported

Explanation: ERROR — The program contains a reference to a built-in function not supported by this version of BASIC.

User Action: Remove the function reference.

built in function requires numeric expression

Explanation: ERROR — The program specifies a string expression for a built-in function that requires a numeric argument.

User Action: Supply a numeric expression for the built-in function.

Cannot change /DEBUG after LOAD command

Explanation: ERROR — An attempt was made to change the /[NO]DEBUG qualifier after an object module was loaded.

User Action: Use the SCRATCH command to clear memory; change the /[NO]DEBUG qualifier to the desired default; and then recompile and reload the object module so that the loaded object module corresponds with the present environment defaults for /DEBUG.

can't continue

Explanation: ERROR — A CONTINUE command was typed after changes had been made to the source code.

User Action: After changes have been made to the source code, you can only run the program; you cannot continue it.

can't find psect <name>

Explanation: FATAL — An internal error occurred when the BASIC LOAD or RUN command was executed, indicating that the compiler could not find a MAP or compiler PSECT.

User Action: Submit an SPR including all relevant information.

can't GOTO outside current procedure

Explanation: WARNING — The target line number of an immediate mode GOTO statement is outside of the currently compiled procedure.

User Action: None. If you run a source file containing more than one program unit, the currently compiled program is the last program unit in the source file. If you use the OLD command to place a program in memory, load one or more object modules, and then type RUN, the currently compiled procedure is the program you read into memory with the OLD command.

CHAIN does not support line-number clause

Explanation: ERROR — A CHAIN statement contains a LINE keyword and a line-number argument.

User Action: Remove the LINE keyword and the line-number argument.

CHANGE statement is ambiguous

Explanation: ERROR — A string variable and a numeric array have the same name in a CHANGE statement.

User Action: Change the name of the string variable or the numeric array.

CHANGES not allowed on primary key

Explanation: ERROR — The PRIMARY KEY clause in an OPEN statement specifies CHANGES.

User Action: Remove the CHANGES keyword; you cannot change the value of a primary key.

channel expression must be numeric

Explanation: ERROR — The program contains a non-numeric channel expression, for example, PUT #A$.

User Action: Change the channel expression to be numeric.

COM/MAP <name> is too large

Explanation: ERROR — The program contains a MAP or COMMON area of storage longer than 32767 bytes.

User Action: Reduce the length of the COMMON or MAP storage.

COMMON/MAP area sizes are not equal for section

Explanation: WARNING — A MAP or COMMON statement with the same name exists in more than one program module, but the size of each area differs.

User Action: Make the size of the COMMON or MAP areas equal in size in all modules.

conflicting data type specifications

Explanation: ERROR — The program contains a declarative statement containing two or more consecutive and contradictory data type keywords, for example, DECLARE REAL BYTE.

User Action: Remove one of the data type keywords or make sure that the keywords refer to the same generic data type; for example, DECLARE REAL SINGLE is valid.

constant <name> not allowed in assignment context

Explanation: ERROR — The program tries to assign a value to a user-defined constant.

User Action: Remove the assignment statement; once you have assigned a value to a declared constant, you cannot change it.

constant expression required

Explanation: ERROR — A statement specifies a variable, built-in function reference or exponentiation where a constant is required.

User Action: Supply an expression containing only literals or declared constants, or remove the exponentiation operation.

constant is inconsistent with the type of <name>

Explanation: ERROR — A DECLARE CONSTANT statement specifies a value that is inconsistent with the data type of the constant, for example, a BYTE value specified for a REAL constant.

User Action: Change the declaration so that the data type of the value matches that of the constant.

current scale factor is <number>

Explanation: WARNING — A SCALE command did not specify a scale factor. The SCALE command is ignored and the present scale factor displayed.

User Action: None.

data type keyword not allowed in SUB statement

Explanation: ERROR — A SUB statement contains a data type keyword between the subprogram name and the parameter list.

User Action: Remove the data type keyword. In a SUB statement, data type keywords can appear only within the parameter list.

data type required for variable <vbl-name> with /EXPLICIT

Explanation: ERROR — A program compiled with the /TYPE:EXPLICIT qualifier declares a variable without specifying a data type.

User Action: Supply a data type keyword for the variable or compile the program without the /TYPE:EXPLICIT qualifier.

data type required in EXTERNAL CONSTANT declaration

Explanation: ERROR — An EXTERNAL CONSTANT statement has no data-type keyword.

User Action: Supply a data-type keyword to specify the data type of the external constant.

DEF <name> mode not as declared

Explanation: ERROR — The specified data type in a function declaration disagrees with the data type specified in the function definition.

User Action: Make the data type specifications match in both the function declaration and the function definition.

DEF <name> not defined

Explanation: ERROR — The program contains a reference to a non-existent user-defined function.

User Action: Define the function in a DEF statement.

DEF invocation not allowed in assignment context

Explanation: ERROR — A DEF function invocation (including a parameter list) appears on the left side of an assignment statement.

User Action: Remove the assignment statement. You cannot assign values to a function invocation.

DEF* formal <formal-name> inconsistent with usage outside DEF*

Explanation: ERROR — A DEF* formal parameter has the same name as a program variable but has different attributes.

User Action: You should not use the same names for DEF* parameters or program variables. If you do, you must make sure that they have the same data type and size.

directive must be only item on line

Explanation: ERROR — The program contains a compiler directive that is not the only item on the line.

User Action: Place the directive on its own line.

directive not valid in immediate mode

Explanation: ERROR — A compiler directive was typed in the BASIC environment.

User Action: None. Compiler directives are invalid in immediate mode.

division by zero

Explanation: WARNING — The value of a number divided by zero is indeterminate.

User Action: Change the expression so that no expression is divided by the constant zero.

DOUBLE constant required

Explanation: ERROR — The program contains a DECLARE DOUBLE CONSTANT statement that specifies an expression for the constant value.

User Action: Remove the expression. You can specify only literal values when declaring floating-point constants.

duplicate OPEN clause

Explanation: ERROR — An OPEN statement contains more than one clause of the same type.

User Action: Remove one of the clauses.

DYNAMIC attribute only valid for MAP areas

Explanation: ERROR — A COMMON keyword is followed by the DYNAMIC keyword.

User Action: Remove the DYNAMIC keyword. The DYNAMIC attribute is valid only for MAP areas.

ELSE appears in improper context, ignored

Explanation: ERROR — The program contains an ELSE clause that either is not preceded by an IF statement or that appears after an IF has been terminated with a line number or END IF statement.

User Action: Remove either (1) the ELSE clause, (2) the terminating line number, or (3) the END IF statement.

END IF appears in improper context, ignored

Explanation: ERROR — The program contains an END IF statement that either is not preceded by an IF statement or occurs after an IF has been terminated by a line number.

User Action: Supply an IF statement or remove the terminating line number.

end of DEF seen while not in DEF

Explanation: ERROR — An FNEND or END DEF statement has no preceding DEF statement.

User Action: Define the function before inserting an END DEF statement, or delete the END DEF statement.

end of FUNCTION while not in FUNCTION

Explanation: ERROR — The program contains a FUNCTIONEND or END FUNCTION statement without an accompanying FUNCTION statement.

User Action: Supply a function subprogram or remove the FUNCTIONEND statement.

end of line does not terminate IFs due to active blocks

Explanation: ERROR — A THEN or ELSE clause contains a loop block, and a line number terminates the IF-THEN-ELSE block before the end of the loop block.

User Action: Make sure that any loop is entirely contained in the THEN or ELSE clause.

end of SUB seen while not in SUB

Explanation: ERROR — A subprogram has a SUBEND or END SUB statement without a preceding SUB statement.

User Action: Supply a SUB statement as the first statement in the subprogram or delete the END SUB or SUBEND statement.

entire array may not be passed BY VALUE

Explanation: ERROR — The program specifies BY VALUE as the passing mechanism for an entire array.

User Action: You cannot pass an entire array BY VALUE. Specify either BY REF or BY DESC.

entire array not allowed in this context

Explanation: ERROR — The program specifies an entire array in a context that permits only array elements, for example, specifying an entire array in a PRINT statement.

User Action: Remove the reference to the entire array and specify individual array elements.

entire virtual array cannot be a parameter

Explanation: ERROR — The program attempts to pass an entire virtual array as a parameter.

User Action: None. You cannot pass an entire virtual array as a parameter.

error deleting <file-name>

Explanation: ERROR — An error was detected in attempting to delete a file.

User Action: Supply a valid file specification, or take corrective action based on the associated message.

error in program name

Explanation: ERROR — The program name is longer than nine characters or contains nonalphanumeric characters.

User Action: Change the program name to be less than or equal to nine characters, and make sure that it contains only letters and digits.

error opening output file <file-name>

Explanation: FATAL — The COMPILE command includes an illegal output file specification, for example, COMPILE $BASIC.

User Action: Specify a valid file name.

error opening file

Explanation: ERROR — The file specified in a %INCLUDE directive could not be opened. This error message is followed by the specific RMS–11 error.

User Action: Take appropriate action based on the associated RMS–11 error.

executable DIMENSION illegal for static array

Explanation: ERROR — A DIMENSION statement names an array already declared with a DECLARE, COMMON, or MAP statement, or one that was declared statically in a previous DIMENSION statement.

User Action: Remove the executable DIMENSION statement, or originally declare the array as executable in a DIMENSION statement.

exit from DEF while not in DEF

Explanation: ERROR — An FNEXIT or EXIT DEF statement has no preceding DEF statement.

User Action: Define the function before inserting an FNEXIT or EXIT DEF statement.

exit from FUNCTION while not in FUNCTION

Explanation: ERROR — An EXIT FUNCTION or FUNCTIONEXIT statement was encountered in a module that is not a FUNCTION subprogram.

User Action: Remove the EXIT FUNCTION or FUNCTIONEXIT statement.

exit from SUB seen while not in SUB

Explanation: ERROR — A program contains an EXIT SUB or SUBEXIT statement with no preceding SUB statement.

User Action: If the program is a subprogram, supply a SUB statement; otherwise, remove the EXIT SUB or SUBEXIT statement.

expecting IF directive

Explanation: ERROR — The program contains a %END directive that is not immediately followed by a %IF directive.

User Action: Supply a %IF directive immediately following the %END directive.

expecting unary operator or legal lexical operand

Explanation: ERROR — A compiler directive contains an invalid lexical expression, for example, %IF *3% %THEN.

User Action: Correct the lexical expression.

explicit declaration of <name> required

Explanation: ERROR — The program is compiled with the /TYPE:EXPLICIT qualifier in effect and the program contains a variable, constant, function, or subprogram that is not explicitly declared.

User Action: Explicitly declare the data type of the variable, constant, function, or subprogram or compile the program without the /TYPE:EXPLICIT qualifier.

expression not allowed in this context

Explanation: ERROR — The program contains an expression in a context that allows only simple variables, array elements, or entire arrays.

User Action: Remove the expression.

expression too complicated

Explanation: The program contains an expression too complicated to compile.

User Action: Rewrite the expression as two or more less complicated expressions.

external globals not allowed–<name>

Explanation: FATAL — A program module being loaded or run with the BASIC LOAD or RUN command references an external variable or constant.

User Action: To reference external variables or constants you must link your program and run it from the system monitor level.

EXTERNAL name too long, truncating to <new-name>

Explanation: ERROR — An EXTERNAL statement names a symbol longer than six characters.

User Action: Shorten the symbol name. External names must be six characters or less.

EXTERNAL STRING variables not supported

Explanation: ERROR — The program contains an EXTERNAL statement that specifies an external string variable.

User Action: Remove or change the EXTERNAL statement. BASIC does not support external string variables.

extra ELSE directive found

Explanation: ERROR — The program contains a %ELSE directive that is not matched with a %IF directive.

User Action: Make sure that each %ELSE directive is preceded by a %IF directive and that each %IF directive contains no more than one %ELSE clause.

extra END IF directive found

Explanation: ERROR — A program unit contains a %END %IF without a preceding %IF directive.

User Action: Supply a %IF for the %END %IF.

extra left parenthesis in expression

Explanation: ERROR — A compiler directive contains a lexical expression with an extra left parenthesis.

User Action: Remove the extra parenthesis.

extra right parenthesis in expression

Explanation: ERROR — A compiler directive contains a lexical expression with an extra right parenthesis.

User Action: Remove the extra parenthesis.

failure in loading object file

Explanation: ERROR — Either an attempt was made to load a non-BASIC object module, or the compiler could not find the object file referenced by a CALL statement or EXTERNAL FUNCTION reference.

User Action: If the object file resides in the VAX–11 Common Run-Time Library, you must link the program at DCL level. If the object file is in a user-supplied library, use the DCL LIBRARY command to make the missing object module available. You can load only BASIC object modules.

FIELD valid only for dynamic string variables

Explanation: ERROR — A FIELD statement contains a numeric or fixed-length string variable.

User Action: Remove the numeric or fixed-length string variable. Only dynamic string variables are valid in FIELD statements.

FIELDed variable <vbl-name> cannot be a parameter

Explanation: ERROR — The parameter list in a reference to a DEF statement or a sub-program contains a string variable or string array element that also appears in a FIELD statement.

User Action: None. If a variable appears in a FIELD statement, the variable cannot be passed as a parameter.

file access error for INCLUDE directive file <file-name>

Explanation: FATAL — The file named in the %INCLUDE directive was correctly opened but could not be read for some reason. For example, the disk drive was switched offline.

User Action: Take action based on the associated RMS–11 error messages.

FILL not allowed in DYNAMIC MAP

Explanation: ERROR — A DYNAMIC MAP statement contains a FILL item.

User Action: Remove the FILL item.

FILL number <n> in overlay <m> of MAP <name> too big

Explanation: ERROR — A FILL string length or repeat count caused the compiler to try to allocate more than 2^{31} longwords of storage.

User Action: Check the specified MAP statement and change the FILL string length or repeat count.

floating point error or overflow

Explanation: WARNING — The program contains a numeric expression whose value is outside the valid range for floating-point numbers.

User Action: Modify the expression so that its value is within the allowable range.

FORM FEED must appear at end of line

Explanation: ERROR — A form feed character is followed by other characters in the same line.

User Action: Remove the characters following the form feed. A form feed must be the last or only character on a line.

formal parameter <name> inconsistent with actual

Explanation: An actual parameter in a DEF function invocation does not agree in data type with the formal parameter in the DEF statement.

User Action: Change the actual parameter in the function invocation to match the data type of the formal parameter in the DEF statement.

formal parameter must be supplied for <name>

Explanation: ERROR — The declaration of a DEF, SUB, or FUNCTION routine contains the parentheses for a parameter list, but no parameters.

User Action: Supply a parameter list or remove the parentheses.

formal string parameters may not be FIELDed

Explanation: ERROR — A variable name appears both in a subprogram formal parameter list and a FIELD statement in the subprogram.

User Action: Remove the variable from the FIELD statement or the parameter list.

found <item> when expecting <item>

Explanation: ERROR — The program contains a syntax error. BASIC displays the item at which the error was detected, then displays one or more items that make more sense in that context. The compilation continues so that other errors may be detected. The actual program line remains unchanged and no object file is produced.

User Action: Examine the line carefully to discover the error. Change the program line to correct the syntax error.

function invocation not allowed in assignment context

Explanation: ERROR — An external function invocation (including a parameter list) appears on the left side of an assignment statement.

User Action: Remove the assignment statement. You cannot assign values to a function invocation.

function nested too deep

Explanation: ERROR — The program contains too many levels of function definitions within function definitions.

User Action: Reduce the number of nested functions.

<name> has a passing mechanism specified with no parameter list

Explanation: ERROR — A CALL statement, external function reference, or EXTERNAL statement specifies a BY clause but does not specify a formal parameter list.

User Action: Remove the BY clause or supply a parameter list.

IDENT directive may appear only once per module

Explanation: WARNING — The program contains more than one %IDENT directive.

User Action: Remove all but one %IDENT directive.

IDENT directive name is too long

Explanation: WARNING — The quoted string in a %IDENT directive is too long.

User Action: Reduce the length of the string. The maximum length is six characters.

IF directive expression must be terminated by THEN

Explanation: ERROR — A %IF directive contains a %ELSE clause with no intervening %THEN clause.

User Action: Insert a %THEN clause.

IF directive in INCLUDE directive needs END IF directive in same file

Explanation: ERROR — A %INCLUDE directive that is modified by a %IF directive is missing a %END %IF directive.

User Action: Supply a %END %IF directive.

<n> IF statement(s) not terminated

Explanation: ERROR — The program contains an IF-THEN-ELSE statement within a block (for example, a FOR-NEXT, SELECT-CASE, or WHILE block), and the end of the block was reached before the IF-THEN-ELSE statement was terminated.

User Action: Check program logic to be sure IF-THEN-ELSE statements are terminated with a line number or an END IF statement before the end of the block is reached.

illegal argument for command

Explanation: ERROR — An argument was entered for a command that does not take an argument, or an invalid argument was entered for a command, for example, SCALE A or LIST A.

User Action: Re-enter the command with the proper arguments.

illegal argument passing mechanism

Explanation: ERROR — The program specifies an invalid argument passing mechanism, for example, passing strings or arrays BY VALUE, or passing an entire virtual array.

User Action: Check all elements for proper parameter passing mechanism.

illegal character <ASCII code>

Explanation: WARNING — The program contains illegal or incorrect characters.

User Action: Examine the program for correct usage of the BASIC character set; possibly delete the character.

illegal character <ASCII code> in external name

Explanation: ERROR — The external symbol in an EXTERNAL FUNCTION or CONSTANT declaration contains an invalid character.

User Action: Remove the invalid character. External names can use only printable ASCII characters. ASCII values are in the range 32 to 126, inclusive.

illegal character <ASCII value> in IDENT directive

Explanation: ERROR — A %IDENT directive contains an illegal character with the reported ASCII value.

User Action: Remove the illegal character.

illegal constant type

Explanation: ERROR — The program contains an invalid declaration, for example, DECLARE RFA CONSTANT.

User Action: Remove the invalid data type. You cannot declare constants of the RFA data type.

illegal I/O channel

Explanation: ERROR — A constant channel expression is greater than 12 or less than zero.

User Action: Change the channel expression to be within the range zero to 12.

illegal library name

Explanation: ERROR — A compiler command or qualifier specifies an invalid library name, for example, DSKLIB NONE.

User Action: Specify a valid library name.

illegal line number

Explanation: ERROR — A line number outside the valid range was typed.

User Action: Enter only line numbers in the range 1 to 32767, inclusive.

illegal line number in CHAIN

Explanation: ERROR — A CHAIN with LINE statement specifies an invalid line number. Either the number is outside the valid range, or a string expression follows the LINE keyword.

User Action: Supply an integer line number between 1 and 32767, inclusive.

illegal loop nesting, expecting NEXT <variable>

Explanation: ERROR — The program contains overlapping loops.

User Action: Examine the program logic to make sure that the FOR and NEXT statements for the inside loop lie entirely within the outside loop.

illegal matrix operation

Explanation: WARNING — The program attempts matrix division. The operation is treated as a MAT multiplication, and the compilation continues.

User Action: Remove the attempted matrix division. BASIC does not support this operation.

illegal mode mixing

Explanation: ERROR — The program contains string and numeric operands in the same operation.

User Action: Change the expression so that it contains either string or numeric operands, but not both.

illegal multiple definition of name <name>

Explanation: ERROR — The program uses the same name for:

- More than one variable
- A variable and a MAP
- A variable and a COMMON
- A MAP and COMMON

User Action: Use unique names for variables, COMMONs, and MAPs.

illegal operation for argument

Explanation: ERROR — The program performs an operation that is inconsistent with the data type of the arguments, for example, an arithmetic operation on variables of the RFA data type.

User Action: Remove the operation or change the data type of the arguments.

illegal string operator

Explanation: ERROR — The program specifies an invalid string operation, for example, A$ = B$ – C$.

User Action: Replace the invalid operator.

illegal switch usage—<text>

Explanation: ERROR — An invalid qualifier was specified with a compiler command, for example, SET/ABC.

User Action: Specify a valid qualifier.

illegal usage of FIELDed variable

Explanation: ERROR — Either (1) a MOVE TO or MOVE FROM statement contains a string variable or string array element that also appears in a FIELD statement, or (2) a MAT statement operates on a string array element that appears in a FIELD statement.

User Action: Either (1) remove the variable from the FIELD statement or the MOVE statement, or (2) remove the array from the MAT statement.

illegal use of unary operator

Explanation: ERROR — A compiler directive contains an invalid lexical expression, for example, %IF 1— 2, or %IF 1 NOT 2.

User Action: Correct the invalid lexical expression.

illegally formed name

Explanation: ERROR — The program contains an invalid user identifier (such as a variable, constant, or function name).

User Action: Change the name to comply with the rules for naming user identifiers. See the *BASIC Reference Manual* for more information.

illegally formed numeric constant

Explanation: ERROR — The program contains either (1) an invalid E-format expression or (2) a numeric constant with a digit that is invalid in the specified radix, for example, a decimal constant containing a hexadecimal digit.

User Action: Supply a valid E-format expression or a constant that is valid in the specified radix.

illegally nested DEFs

Explanation: ERROR — The program contains a DEF function block within another DEF function block.

User Action: Remove the inner DEF block. A DEF block cannot contain another DEF block.

immediate mode operation requires storage allocation

Explanation: ERROR — An immediate mode statement attempted to allocate storage, for example, to create a new variable.

User Action: None. You cannot create new storage in immediate mode.

implicit declaration of <name> illegal in immediate mode

Explanation: ERROR — A new variable was named in an immediate mode statement after a STOP statement, for example, PRINT B after a STOP statement in a program that has no variable named B.

User Action: None. You cannot create new variables in immediate mode after a STOP statement.

implied continuation not allowed

Explanation: ERROR — The program contains an implied continuation line after a statement that does not allow implicit continuation, for example, a REM statement.

User Action: Use an ampersand (&) to continue the statement.

implied declaration not allowed for <name> with /TYPE = EXPLICIT

Explanation: ERROR — A program compiled with the TYPE = EXPLICIT qualifier contains an implicitly declared variable.

User Action: Compile the program without the TYPE = EXPLICIT qualifier, or declare the variable explicitly.

inaccessible code follows line <n> statement <m>

Explanation: WARNING — The program contains one or more statements that cannot be accessed, for example, a multi-statement line whose first statement is GOTO, EXIT, ITERATE, RESUME, or RETURN.

User Action: Make sure that the GOTO, EXIT, ITERATE, RESUME, or RETURN statement is the only statement on a numbered line or the last statement on a multi-statement line.

INCLUDE directive file must be on a random access device

Explanation: ERROR — A %INCLUDE directive specifies a device other than a disk.

User Action: Change the %INCLUDE directive to specify a random access device.

INCLUDE directive RMS error number <number>

Explanation: ERROR — A %INCLUDE directive caused an RMS–11 error when accessing the specified file.

User Action: Take action based on the reported RMS–11 error number.

INCLUDE directive syntax error

Explanation: ERROR — A %INCLUDE directive is not followed by a quoted string.

User Action: Supply a quoted string.

inconsistent function usage for function <name>

Explanation: ERROR — The parameter list in a DEF function invocation contains a string where the function expected a number, or vice versa. This message is issued only when the invocation occurs before the DEF statement in the program.

User Action: Supply a correct parameter in the function invocation or correct the parameter list in the DEF statement.

inconsistent subscript use for <array-name>

Explanation: ERROR — The number of subscripts in an array reference does not match the number of subscripts specified when the array was created.

User Action: Specify the same number of subscripts.

input prompt must be a string constant

Explanation: ERROR — An INPUT, LINPUT, or INPUT LINE argument list contains a numeric constant immediately following the statement.

User Action: Remove the numeric constant. You can specify only a string constant immediately after an INPUT, LINPUT, or INPUT LINE statement.

insufficient space for MAP DYNAMIC variable in MAP <name>

Explanation: ERROR — A variable named in a MAP DYNAMIC statement is larger than in the MAP statement, for example, an HFLOAT variable in a MAP DYNAMIC statement that is only four bytes long in a MAP statement.

User Action: Increase the size of the MAP so it is large enough to hold the largest member.

integer constant exceeds machine integer size

Explanation: ERROR — The value specified in a DECLARE CONSTANT statement exceeds the largest allowable value for an integer. The maximum value is 32767.

User Action: Supply a value in the valid range.

integer constant required

Explanation: ERROR — The program contains a non-integer named constant in a context that requires an integer. For example:

DIM A ('123'D)

User Action: Supply an integer constant.

integer error or overflow

Explanation: WARNING — The program contains an integer expression whose value is outside the valid range.

User Action: Modify the expression so that its value is within the allowable range, or use an integer data type that can contain all possible values for the expression.

internal logic error detected

Explanation: ERROR — An internal logic error was detected.

User Action: This error should never occur. Submit a Software Performance Report with a machine-readable copy of the source program.

invalid conversion requested

Explanation: ERROR — The program contains a reference to the REAL or INTEGER function, and the argument to the function is an entire array or an RFA expression.

User Action: Remove the invalid argument. The argument to these functions must be a numeric expression.

invalid integer type

Explanation: ERROR — A reference to the INTEGER function contains an invalid data type keyword, for example, A = INTEGER(A, SINGLE).

User Action: Change the invalid data type keyword. The INTEGER function returns only BYTE, WORD, or LONG values.

invalid real type

Explanation: ERROR — A reference to the REAL function contains an invalid data type keyword, for example, A = REAL(A, LONG).

User Action: Change the invalid data type keyword. The REAL function returns only SINGLE, DOUBLE, GFLOAT, or HFLOAT values.

<command> is an illegal command from initialization file

Explanation: ERROR — An initialization file contains an invalid command, such as COMPILE.

User Action: Remove the invalid command.

<text> is an invalid keyword value

Explanation: FATAL — The command supplied an invalid value for a keyword.

User Action: Supply a valid value.

<clause> is an unsupported OPEN clause

Explanation: ERROR — An OPEN statement specifies invalid attributes for the file.

User Action: Substitute valid attributes for the file.

<name> is not a DYNAMIC MAP variable of MAP <name>

Explanation: ERROR — A REMAP statement names a variable that was not named in the MAP DYNAMIC statement for the associated MAP statement.

User Action: Remove the variable from the REMAP statement, or name the variable in the MAP DYNAMIC statement for the associated MAP statement.

ITERATE must appear within a loop

Explanation: ERROR — The program contains an ITERATE statement that is not within a FOR-NEXT, WHILE, or UNTIL loop.

User Action: Remove the ITERATE statement or surround it with a loop.

jump into DEF

Explanation: ERROR — The program attempts to transfer control into a DEF block.

User Action: Change the control statement; you cannot transfer control into a DEF block except by invoking the function.

jump out of DEF

Explanation: ERROR — The program attempts to transfer control out of a DEF block.

User Action: Change the control statement; you cannot transfer control out of a DEF block except by an EXIT DEF, FNEXIT, FNEND, or END DEF statement.

jump out of program unit

Explanation: ERROR — In a source file containing more than one program module, a statement attempts to transfer control from one module into another.

User Action: Change the statement that attempts to transfer control; you cannot transfer control into a different program module.

jump to label: <label> is into a block

Explanation: ERROR — The program attempted to transfer control into a FOR-NEXT, WHILE, UNTIL, or SELECT-CASE block.

User Action: Change the program logic so that it does not transfer control into a block.

jump to line number <number> is into a block

Explanation: INFORMATION — The program transfers control to a line number within a FOR-NEXT, WHILE, UNTIL, or SELECT-CASE block.

User Action: This is an informational message. However, it is bad programming practice to transfer control into a block.

jump to unreferencable line number <lin-num>

Explanation: ERROR — A RESUME, GOSUB, or GOTO statement attempts to transfer control to a CASE statement.

User Action: Label or number the SELECT statement and transfer control to the beginning of the SELECT-CASE block.

key <vbl-name> in MAP <map-name> cannot be a dynamic variable

Explanation: ERROR — A KEY clause in an OPEN statement specifies a variable declared as dynamic in a MAP DYNAMIC statement.

User Action: Specify a static variable in the KEY clause; that is, declare the variable in a MAP statement, not a MAP DYNAMIC statement.

KEY <vbl-name> in MAP <name> is too long (max is 255)

Explanation: ERROR — A KEY variable is longer than 255 characters.

User Action: Reduce the length of the KEY variable. The maximum key length is 255 characters.

KEY <vbl-name> is not an unsubscripted variable in MAP <name>

Explanation: ERROR — An OPEN statement for an indexed file specifies a KEY variable that does not appear in a MAP statement.

User Action: Place the KEY variable in the MAP statement referenced by the OPEN statement's MAP clause.

KEY clauses require a MAP clause

Explanation: ERROR — An OPEN statement specifies KEY clauses without specifying a MAP clause.

User Action: Supply a MAP clause to define the position of the keys in the record buffer.

key is needed for indexed files

Explanation: ERROR — The program attempts to open an indexed file for output, and the PRIMARY KEY clause is missing.

User Action: Supply a PRIMARY KEY clause.

key must be either integer or string

Explanation: ERROR — A FIND or GET statement on an indexed file contains a key specification that is not an integer or string.

User Action: Change the key specification to be an integer or a string.

key segment <vbl-name> in map <map-name> must be a string key

Explanation: ERROR — An OPEN statement specifies a segmented key containing a numeric variable. For example:

```
10   OPEN ''INDEX.DAT'' AS FILE #1, ORGANIZATION INDEXED,  &
          PRIMARY KEY (A$, B$, C%), MAP ABC
```

User Action: Specify only string variables in segmented keys.

key, <vbl-name> in map <map-name> must be either integer or string

Explanation: ERROR — An OPEN statement contains a key specification that is not an unsubscripted integer or string variable.

User Action: Change the key specification to be an unsubscripted integer or string variable.

<keyword> keyword inconsistent with <keyword>

Explanation: ERROR — An OPEN statement contains contradictory record format specifications, for example, both FIXED and VARIABLE.

User Action: Specify only one record format.

<keyword> keyword is inconsistent with file organization

Explanation: ERROR — An OPEN statement contains a keyword that is inapplicable to the file organization, for example, ACCESS SCRATCH with ORGANIZATION VIRTUAL.

User Action: Remove the inconsistent keyword.

keyword inconsistent with <OPEN clause> clause

Explanation: ERROR — An OPEN statement contains an ALLOW, ACCESS, or RECORDTYPE clause whose keyword argument is invalid, for example, ACCESS FORTRAN.

User Action: Change the clause argument to a valid keyword for that clause.

<text> keyword requires a value

Explanation: ERROR — A keyword command was typed without a value.

User Action: Supply a valid keyword value.

keyword requires INDEXED or RELATIVE organization

Explanation: ERROR — An OPEN statement specifies a BUCKETSIZE or BUFFER clause for a sequential file.

User Action: Remove the BUCKETSIZE or BUFFER clause; they are valid only on relative or indexed files.

keyword requires INDEXED organization

Explanation: ERROR — An OPEN statement specifies a CONNECT, PRIMARY, or ALTERNATE KEY clause for a relative or sequential file.

User Action: Remove the CONNECT or KEY clause; they are valid only on indexed files.

keyword requires SEQUENTIAL organization

Explanation: ERROR — An OPEN statement specifies a BLOCKSIZE, NOREWIND, or NOSPAN clause for a relative or indexed file.

User Action: Remove the BLOCKSIZE, NOREWIND, or NOSPAN clause; they are valid only for sequential files.

label <label> not defined

Explanation: ERROR — The program tries to transfer control to a non-existent label.

User Action: Define the label before transferring control to it.

label <name> does not label an active block statement

Explanation: ERROR — An EXIT statement in a loop, IF-THEN-ELSE, or SELECT-CASE block specifies a label which does not refer to that block.

User Action: Change the program so that the label actually refers to the block in which the EXIT statement occurs.

label <name> does not label an active loop statement

Explanation: ERROR — In a loop, an EXIT or ITERATE statement specifies a label which does not refer to that loop.

User Action: Change the program so that the label actually refers to the loop in which the EXIT or ITERATE statement occurs.

label not allowed on RESUME

Explanation: ERROR — A RESUME statement specifies a label rather than a line number.

User Action: Change the label to a line number.

language feature is declining

Explanation: INFORMATION — This error is reported only when the FLAG:DECLINING qualifier is in effect. The program contains a language feature that is not recommended for new program development, for example, the FIELD statement.

User Action: DIGITAL suggests that you use (1) MAP, MAP DYNAMIC and REMAP statements instead of a FIELD statement, (2) EDIT$ rather than CVT$$ functions, and (3) overlaid MAPs rather than CVTxx functions.

language feature is operating system dependent

Explanation: ERROR — The program contains a PRINT statement with RECORD clause on a system which does not support the RECORD clause.

User Action: Remove the RECORD clause.

LET directive syntax error

Explanation: ERROR — A %LET directive contains a syntax error, for example, an invalid lexical identifier.

User Action: Use the correct syntax for the %LET directive.

lexical identifier must be declared before reference

Explanation: ERROR — A %IF directive names a lexical constant which was not named in a preceding %LET directive.

User Action: Declare the lexical constant with the %LET directive before referencing it.

line number <n> undefined due to conditional compilation

Explanation: ERROR — The program references a line number which does not appear in the object code as a result of the branch taken in a %IF-%THEN-%ELSE-%END-%IF directive.

User Action: Change the %IF-%THEN-%ELSE-%END-%IF directive or remove the line number reference.

line number may not appear in INCLUDE directive file

Explanation: ERROR — The file specified in a %INCLUDE directive contains a line number.

User Action: Remove the line number from the file.

line too long

Explanation: ERROR—A program contains either (1) a text line that exceeds the maximum characters allowed (the maximum length of a text line is 225 characters on RSTS/E systems and 132 characters on RSX–11M/M-PLUS systems), or (2) a multi-statement line that contains more than 32767 characters.

User Action: Either (1) break the line into two text lines, using the ampersand (&) continuation character, or (2) split the multi-statement line by adding a new line number.

logical operation on non-integer quantity

Explanation: ERROR—The program contains a logical operation performed on strings or real numbers.

User Action: Change the logical operands to integers.

loop control variable must be a numeric variable

Explanation: ERROR—A FOR statement attempts to assign a string expression as the loop control variable's initial value.

User Action: Remove the string expression. You can assign only numeric values as the loop's initial value.

loop initial value must be a numeric expression

Explanation: ERROR—A FOR statement attempts to assign a string value to the loop control variable.

User Action: Remove the string expression. You can assign only numeric values to the loop control variable.

loop limit must be numeric

Explanation: ERROR—A FOR statement attempts to assign a string expression as the loop control variable's limiting value.

User Action: Remove the string expression. You can assign only numeric values as the loop control variable's limiting value.

loop will never execute

Explanation: WARNING—The program contains a FOR-NEXT loop that is not executable; for example, FOR I% = 1% TO 0%. Compilation continues, but the loop is ignored.

User Action: Change the loop parameters or insert an appropriate STEP clause.

MAP <name> larger than previously defined

Explanation: FATAL—When a program is run with the BASIC RUN command, the length of a map is defined by the length of the first occurrence of the map. Therefore, if the current program module in memory defines a MAP as 100 bytes in length and a loaded program module defines the same MAP as 50 bytes in length, the MAP is defined as containing 50 bytes (because the loaded module was compiled before the program module currently in memory).

User Action: Make sure the MAP statements are the same size.

MAP <name> used in OPEN not defined

Explanation: ERROR — An OPEN statement's MAP clause references a non-existent MAP statement.

User Action: Define the MAP statement referenced by the MAP clause, or remove the MAP clause.

MAP DYNAMIC <map-name> may not be larger than 32767 bytes

Explanation: ERROR — A MAP DYNAMIC statement references a map that is greater than 32767 bytes in size.

User Action: Reduce the size of the map, as defined in the MAP statement(s), to 32767 bytes or less.

MAP DYNAMIC <name> requires corresponding static MAP

Explanation: ERROR — The program contains a MAP DYNAMIC statement whose MAP name does not appear in a MAP statement.

User Action: Provide a MAP statement with the same name as the MAP DYNAMIC name.

MAP statement requires map name

Explanation: ERROR — A MAP statement does not specify a map name.

User Action: Specify a name for the MAP.

MAP too large in OPEN

Explanation: ERROR — The size of the MAP area referenced in an OPEN statement is greater than 32767 bytes.

User Action: Reduce the size of the MAP area.

MAP variable <name> referenced before declaration

Explanation: INFORMATION — A reference to a MAP variable occurs before the MAP statement.

User Action: Make sure that the MAP statement precedes any references to variables in the MAP.

MAT statements require one or two dimensions

Explanation: ERROR — A MAT statement references an array of more than two dimensions.

User Action: Remove the array reference. MAT statements are valid only on arrays of one or two dimensions.

matrix dimension error

Explanation: ERROR—The program either (1) contains a MAT IDN, MAT TRN, or MAT INV performed on a one dimensional array, or (2) performs a matrix operation which requires identical subscripts in the operand arrays and those arrays have different subscripts.

User Action: Dimension the arrays to the proper number of subscripts.

maximum conditional compilation depth exceeded

Explanation: ERROR — There are too many nested %IF-%THEN-%ELSE-%END-%IF directives in the program.

User Action: Reduce the number of nested %IF-%THEN-%ELSE-%END-%IF directives.

maximum number of dimensions exceeded. Maximum is <number>

Explanation: ERROR — An array declaration specifies more than the allowed number of dimensions.

User Action: Reduce the number of dimensions. The maximum is 8.

maximum parameters exceeded for <name>. Maximum is <number>

Explanation: ERROR — The program attempts to declare a DEF statement with more than 8 parameters or a subprogram with more than 255 parameters.

User Action: Reduce the number of parameters; DEF statements allow up to 8 parameters and subprograms allow up to 255 parameters.

<item> may not be passed BY <mechanism>

Explanation: ERROR — The program specifies an incorrect passing mechanism for a parameter's data type or an invalid parameter. For example, you cannot pass an entire array BY VALUE, nor can you pass a label as a parameter.

User Action: Specify a valid parameter or passing mechanism.

mismatched END, expected <block>

Explanation: ERROR — The program contains an incorrect END statement, for example, an END RECORD statement instead of an END GROUP statement.

User Action: Supply the correct type of END statement.

missing END IF directive before end of program unit

Explanation: ERROR — A %IF directive crosses a program module boundary.

User Action: Terminate the %IF with a %END %IF before beginning a new source module.

mode for parameter <n> of routine <name> changed to match declaration

Explanation: ERROR — The data type specified in a routine invocation does not match that of the routine declaration. BASIC issues this message only if the data type conversion results in a parameter that cannot be modified by the routine that was invoked.

User Action: Make the data type specifications in the declaration and the invocation match.

mode for parameter <n> of routine <name> not as declared

Explanation: ERROR — The CALL or invocation of a routine specifies a string argument for a parameter that was specified as a numeric when the routine was declared, or vice versa.

User Action: Change the string parameter to numeric, or vice versa.

module <name> not a BASIC-PLUS–2 object module

Explanation: FATAL — A program module being loaded with the BASIC LOAD command was not created by BP2 V2.

User Action: If the module source file is written in an earlier version of BP2, compile the source file with the BASIC COMPILE command and reload the module. If the source file was written in another language (MACRO, for example), link and run your program from the system monitor level.

more than one main module

Explanation: FATAL — There is more than one main program loaded, or a main module is loaded and another main module is the current program module when the RUN command is given.

User Action: Reload the program modules, removing all but one of the main modules.

multiple definition of <name>

Explanation: ERROR — A variable is declared in more than one declarative statement.

User Action: Make sure that the variable is declared only once.

multiple definition of lexical identifier is illegal

Explanation: ERROR — A lexical constant is named in more than one %LET directive.

User Action: Declare the lexical constant only once with %LET.

name is too long, changed to <name>

Explanation: WARNING — A variable or array name is longer than 31 characters. BASIC truncates the name to 31 characters and continues compilation so that other errors may be detected. The actual program line remains unchanged and no object file is produced.

User Action: Reduce the length of the variable name to 31 or fewer characters.

named array <array-name> is too large

Explanation: ERROR — An array requires more than $2^{29}-1$ bytes of storage.

User Action: Reduce the size of the array.

negative FILL or string length

Explanation: ERROR — The program contains a negative FILL specification or string length.

User Action: Change the FILL specification or string length to a positive number.

nested FOR loops with same control variable <name>

Explanation: ERROR — The program contains nested FOR-NEXT loops that use the same index variable.

User Action: Change the index variable for all but one of the loops.

no change made

Explanation: WARNING — The search string in an EDIT command was not located in the text.

User Action: Enter valid search string.

no file specified for command in initialization file—command ignored

Explanation: WARNING — An initialization file contains a BRLRES, DSKLIB, LIBRARY, ODLRMS, or RMSRES command or qualifier without the required file specification.

User Action: Provide a valid file specification.

no descriptor allocated for array <name>

Explanation: ERROR — An immediate mode statement required an array descriptor, but it was not available. BASIC allocates array descriptors only if the program code requires it.

User Action: None.

no main program

Explanation: FATAL — When the BASIC RUN command was given, at least one subprogram was loaded or currently in memory, but there was no main module loaded as an object module or currently in memory.

User Action: If there is no current program module in memory, call in the main module with the OLD command. Otherwise, compile and reload all subprograms as object modules and then call the main program into memory.

no such MAP area <name>

Explanation: ERROR — A REMAP statement names a non-existent MAP area.

User Action: Supply a MAP statement before executing the REMAP statement.

non-continued statement has no line number near <lin-num>

Explanation: ERROR — A new line in the source file (1) does not follow a line ending with an ampersand, (2) does not begin with a line number, and (3) does not start with a space or tab (specifying an automatic continuation line.)

User Action: Start automatic continuation lines with a space or tab, use an ampersand as the last character of the preceding line, or start the line with a line number.

numeric array expected

Explanation: ERROR — A CHANGE statement does not specify a numeric array.

User Action: Supply a numeric array in the CHANGE statement.

numeric constant required

Explanation: ERROR — The program contains a string in a context that requires a numeric constant. For example:

```
DECLARE INTEGER CONSTANT A = "ABC"
```

User Action: Supply a numeric constant.

numeric expression is needed

Explanation: ERROR—The program contains a string expression in a context that requires a numeric expression, for example, WHILE A$.

User Action: Supply a numeric expression.

numeric expression is needed in built-in function

Explanation: ERROR—A reference to a BASIC built-in function contains a string instead of a numeric expression.

User Action: Supply a numeric expression.

OPEN clause <clause> value greater than <number>

Explanation: ERROR — An OPEN statement contains a RECORDSIZE, FILESIZE, EXTENDSIZE, WINDOWSIZE, BLOCKSIZE, BUCKETSIZE, or BUFFER clause whose argument is too large.

User Action: Supply a smaller value for the argument.

operator expected, not found

Explanation: A compiler directive contains an invalid lexical expression which has a right parenthesis immediately followed by a lexical identifier.

User Action: Correct the lexical expression.

operator must follow right parenthesis

Explanation: ERROR — The program contains an incorrect lexical expression.

User Action: Correct the lexical expression.

OPTION clause contradicts prior clause

Explanation: ERROR — The OPTION statement contains contradictory clauses, for example, specifying the default integer size as both BYTE and LONG.

User Action: Remove one of the clauses.

OPTION statement out of sequence

Explanation: ERROR — The OPTION statement is either (1) not the first statement in a main program or (2) not the first statement following the SUB or FUNCTION statement.

User Action: Move the OPTION statement so it is either the first statement in the main program or the first statement following the SUB or FUNCTION statement in the subprogram.

ORGANIZATION UNDEFINED requires FOR INPUT clause

Explanation: ERROR — The program opens a file with ORGANIZATION UNDEFINED, but does not specify FOR INPUT.

User Action: Specify FOR INPUT in the OPEN statement. You cannot write to a file with an undefined file organization. However, once you interpret the file organization using the FSP$ function, you can perform output by closing the file and then reopening it FOR OUTPUT, specifying the appropriate organization clause.

<keyword> overrides NOLINE

Explanation: WARNING — The program (1) was compiled with the NOLINES qualifier and (2) uses a keyword that requires line number information. For example, ERL and RESUME with line number statements both require that the program be compiled with the LINES qualifier.

User Action: None. If you use a keyword that requires line number information, BASIC automatically overrides the NOLINE default and sets LINES in effect.

parameter <n> of <type> structure not as declared

Explanation: ERROR — The actual parameter list in a CALL to a SUB subprogram or an invocation of a FUNCTION subprogram specifies an entire array where the subprogram declaration specified a simple variable, or vice versa.

User Action: Change the actual parameter list to match the declared parameter list, or vice versa.

parameter may not be received by <mechanism>

Explanation: ERROR — The subprogram specifies an incorrect passing mechanism for a parameter's data type or an invalid parameter. For example, you cannot receive an entire array BY VALUE.

User Action: Specify a valid parameter or passing mechanism.

parameter type specification required with /EXPLICIT

Explanation: ERROR — In a program compiled with the /TYPE:EXPLICIT qualifier, no data type keyword is specified for a parameter.

User Action: Supply a data type keyword for the parameter. There are no default data types when you compile a program with the /TYPE:EXPLICIT qualifier.

<n> parameters expected for <routine>

Explanation: ERROR — The CALL statement or invocation of a routine specifies a different number of parameters than the number specified when the routine was declared.

User Action: Change the number of parameters to match the number declared.

passing mechanism disagrees with declaration

Explanation: ERROR — The CALL statement or invocation of a routine specifies a different passing mechanism for a parameter than that specified when the routine was declared.

User Action: Remove the BY clause specified in the CALL statement or invocation; BASIC automatically passes parameters with the passing mechanism specified when the routine was declared.

passing mechanism not allowed for <item>

Explanation: ERROR — The program specifies a passing mechanism in a context other than an external subprogram invocation or declaration.

User Action: Remove the passing mechanism.

passing mechanism not allowed for DEF <vbl-name>

Explanation: ERROR — A DEF invocation specifies a passing mechanism for a parameter.

User Action: Remove the passing mechanism.

PRINT USING clause must be a string expression

Explanation: ERROR — A PRINT USING statement specifies a numeric format string.

User Action: Supply a valid format string.

PRINT USING conflicts with RECORD clause

Explanation: ERROR — A PRINT USING statement contains a RECORD clause.

User Action: Remove the RECORD clause or use the PRINT statement instead of PRINT USING.

program structures nested too deeply

Explanation: FATAL — The program contains too many nested block constructs, for example, DEF function definitions.

User Action: Reduce the number of nested block constructs.

program too big to compile

Explanation: FATAL — The program is too big.

User Action: Recode the program as two or more modules.

program too large to RUN

Explanation: FATAL — Maximum memory has been exceeded by a program being run in the BASIC environment.

User Action: To reduce the memory requirements of your program, write an overlay descriptor (ODL) file to overlay the program modules; then link and run your task from the system monitor level. (If you ran the program with the /DEBUG qualifier, the debugger added to the size of your task. You may be able to run the program in the BASIC environment with the RUN/NODEBUG command.)

radix not supported

Explanation: ERROR — A literal constant specifies a radix. For example, in the following DECLARE statement, H is an invalid radix specifier:

```
10    DECLARE LONG CONSTANT A = H"111"
```

User Action: Remove the radix specifier. All literal constants are decimal in BASIC–PLUS–2.

READ access inconsistent with FOR OUTPUT

Explanation: ERROR — An OPEN statement specifies the FOR OUTPUT and ACCESS READ clauses.

User Action: The FOR OUTPUT clause specifies that a new file is created; ACCESS READ specifies that the program can only read the file. If you want to create a new file, remove the ACCESS READ clause; if you want read-only access to a file, specify the FOR INPUT clause.

READ without DATA statement

Explanation: ERROR — The program contains a READ statement and there are no DATA statements.

User Action: Supply a DATA statement, or remove the READ statement.

real constant expressions not supported

Explanation: ERROR — The program contains a DECLARE REAL CONSTANT statement that specifies an expression for the constant value.

User Action: Remove the expression. You can specify only literal values when declaring floating-point constants.

record too big from INCLUDE directive file

Explanation: FATAL — The file specified in a %INCLUDE directive contains a record longer than 255 characters.

User Action: Edit the file to remove any records longer than 255 characters.

repeat count must be positive numeric

Explanation: ERROR — A FILL item specifies a non-numeric or negative repeat count, for example, FILL(A$) or FILL(–3).

User Action: Supply a valid repeat count.

<item> requires a numeric expression

Explanation: ERROR — The program contains a string expression in a context requiring a numeric expression.

User Action: Supply a numeric expression.

<item> requires conditional expression

Explanation: ERROR — A CASE or IF keyword is immediately followed by a floating-point or string expression.

User Action: Supply a conditional expression (relational, logical, or integer).

<item> requires string expression

Explanation: ERROR — The program contains a numeric expression in a context requiring a string expression, for example, the file specification in an OPEN statement, or the default file specification in a DEFAULTNAME clause.

User Action: Supply a string expression.

result attributes inconsistent with prior declaration

Explanation: ERROR — An external or DEF function declaration specifies a different data type for the function's result than the DEF or FUNCTION statement specifies.

User Action: Change the specified data type in either the declaration or the DEF or FUNCTION statement so that the data types agree.

RFA expression required

Explanation: ERROR — A GET statement's RFA clause contains an expression that is not of the RFA data type.

User Action: Supply a valid RFA expression.

RFA not allowed in this context

Explanation: ERROR — The program attempts to use an RFA expression in an arithmetic expression or other invalid context.

User Action: Remove the RFA expression. You can use the RFA data type only in file I/O, in an assignment statement or in a comparison.

scale factor has been set to <number>

Explanation: WARNING — A SCALE command has reset the scale factor.

User Action: None.

scale factor of <number> is out of range

Explanation: ERROR — The SCALE command specifies a scale factor that is not between 0 and 6, inclusive.

User Action: Supply a valid scale factor.

scale factor out of range–ignored

Explanation: WARNING — The SCALE qualifier specifies a scale factor that is not between 0 and 6, inclusive.

User Action: Supply a valid scale factor.

scale has been truncated to <number>

Explanation: WARNING—A floating-point number was specified in the SCALE command. The number has been truncated and the resulting integer is now the scale factor.

User Action: None.

SCALE is out of range. Valid is 0 to 6.

Explanation: ERROR—The OPTION statement specifies a scale factor that is not between 0 and 6, inclusive.

User Action: Supply a valid scale factor.

scale factor used is 0 for single precision

Explanation: WARNING — An attempt was made to set the SCALE factor while in single precision.

User Action: Set the precision to DOUBLE. You cannot use scaling when in single precision.

SINGLE constant required

Explanation: ERROR — The program contains a DECLARE SINGLE CONSTANT statement that specifies an expression for the constant value.

User Action: Remove the expression. You can specify only literal values when declaring floating-point constants.

SPAN is inconsistent with NOSPAN

Explanation: WARNING—An OPEN statement specifies both SPAN and NOSPAN clauses.

User Action: Remove one of the clauses.

specified numeric exceeds valid character code

Explanation: FATAL — A quoted literal of type character C contains a value outside the valid range, for example, '300'C.

User Action: Use a valid ASCII value.

star (*) is needed in DEF, not "/"

Explanation: ERROR—The program contains a statement that starts with DEF/.

User Action: Change the DEF/ to DEF*.

string constant expression is too long

Explanation: ERROR—The program contains a DECLARE STRING CONSTANT statement where the value assigned to the constant exceeds the maximum number of characters allowed for string constant expressions. In BASIC–PLUS–2, the maximum length of a string constant expression at compile time is 128 characters.

User Action: Change the string constant to a string variable and assign the string expression to the variable at run time.

string constant required

Explanation: ERROR — The program contains a numeric expression in a context that requires a string expression. For example:

```
DECLARE STRING CONSTANT ABC = 123
```

User Action: Supply a string literal or a named string constant.

string expression is needed

Explanation: ERROR — The program contains a numeric expression where a string expression is needed, for example, NAME 1% AS "ABC.DAT".

User Action: Supply a string expression.

string expression is needed in built-in function

Explanation: ERROR — The program specifies a numeric expression for a built-in function that requires a string argument.

User Action: Supply a string expression for the built-in function.

string is too large

Explanation: ERROR — A string exceeds the maximum allowable length. The maximum length is 32767 characters.

User Action: Reduce the length of the string.

string length not allowed on dynamic string <name>

Explanation: ERROR — The program contains a dynamic string variable declaration that specifies a string length.

User Action: Length specifications are allowed only for fixed-length strings; remove the length specification from the dynamic string, or allocate the string in a MAP or COMMON statement.

string length not allowed on MAP DYNAMIC variable

Explanation: ERROR — A string variable in a MAP DYNAMIC statement specifies a string length.

User Action: Remove the string length. All string variables named in a MAP DYNAMIC statement have a length of zero until a REMAP statement executes.

string length not allowed on numeric FILL

Explanation: ERROR — The program contains a numeric FILL item that specifies a length.

User Action: Remove the length specification from the numeric FILL item.

string length not allowed on numeric variable <name>

Explanation: ERROR — The declaration for a numeric variable contains a specification for string length.

User Action: Remove the string length specification.

string length specification for <name> must be numeric

Explanation: ERROR — The length specification for a fixed-length string is non-numeric, for example COMMON A$ = "ABC".

User Action: Supply a numeric length specification.

string literal required for compiler directive

Explanation: ERROR — A quoted string is missing in a compiler directive that requires one, for example, %IDENT.

User Action: Supply a string literal for the compiler directive.

string variable expected

Explanation: ERROR — A CHANGE statement specifies a numeric variable.

User Action: Supply a string variable; the CHANGE statement changes a string variable to a numeric array, and vice versa.

string variable required

Explanation: ERROR — A statement references a numeric variable instead of a string variable, for example LINPUT A%.

User Action: Supply a string variable instead of a numeric variable.

subscript may not be specified for entire array

Explanation: ERROR — A CALL statement or external function reference passes an entire array as a parameter and contains a subscript expression, for example A(,,3).

User Action: Remove the subscript expression. You cannot specify any subscripts when passing an entire array as a parameter.

subscript out of range for <array-name>

Explanation: ERROR — The program references an array element with constant subscript(s) outside the bounds of the array.

User Action: Check program logic to make sure all subscripts are within the bounds of the array.

suffix not allowed on FILL after data-type keyword

Explanation: ERROR — A FILL item defined with an explicit data type ends in a percent or dollar sign.

User Action: Remove the FILL item's percent or dollar sign.

suffix not allowed on variable <name>

Explanation: ERROR — A variable defined with explicit data type ends in a percent or dollar sign.

User Action: Remove the variable's percent or dollar sign.

symbol <name> multiply defined

Explanation: FATAL — More than one subprogram with the same name has been loaded with the BASIC LOAD command.

User Action: Remove or rename the subprograms.

system commands cannot be executed from ini file

Explanation: ERROR — An initialization file contains a command preceded by a dollar sign.

User Action: Remove the command from the initialization file. Initialization files cannot perform system commands.

text following END ignored

Explanation: ERROR — The compiler detected text following an END, END SUB, or END FUNCTION statement.

User Action: Remove the text. In BASIC-PLUS–2, if an END, END SUB, or END FUNCTION statement appears in the program, it must be the last statement.

THEN directive must follow a lexical expression

Explanation: ERROR — A %IF directive contains a lexical expression that is not immediately followed by a %THEN.

User Action: Supply a %THEN clause. %THEN, %ELSE, and %END %IF are required in a %IF directive.

too few arguments

Explanation: ERROR — The invocation of a BASIC built-in function contains too few arguments.

User Action: Supply the correct number of arguments to the function.

too many arguments

Explanation: ERROR — The invocation of a BASIC built-in function contains too many arguments.

User Action: Supply the correct number of arguments to the function.

too many array indices active

Explanation: ERROR — A subscript expression contains more than 100 array indices between the open parenthesis and the close parenthesis.

User Action: Reduce the number of active array indices.

too many function parameters active

Explanation: ERROR — An external function invocation contains too many expressions in the actual parameter list.

User Action: Reduce the number of expressions in the actual parameter by assigning the expressions to temporary variables.

too many keys–limit is 255

Explanation: ERROR — An OPEN statement specifies more than 255 index keys.

User Action: Reduce the number of index keys. The maximum is 255.

too many subprograms or maps

Explanation: FATAL — A program being run in the BASIC environment contains more than 16 MAPs or calls more than 16 subprograms.

User Action: Link and run your program from the system monitor level.

too many temporaries generated for DEF at line <line-number>statement <statement-number>

Explanation: ERROR — A program contains code within a DEF function that contains too large a number of temporary variables.

User Action: Either (1) change the DEF function to an external function, or (2) reduce the number of temporaries required within the DEF function.

TYPE default of STRING is not allowed

Explanation: ERROR — STRING was specified as the default data type in (1) a compiler command, (2) a qualifier to the BASIC DCL command, or (3) an OPTION statement.

User Action: Specify a numeric data type as the default.

unable to copy file, RMS error <error number>

Explanation: FATAL — An RMS–11 error occurred while attempting to copy the RUN task into the user's account. See the *RMS–11 User's Guide* for an explanation of the error.

User Action: Take action based on the associated RMS error.

unaligned COMMON or MAP variable <vbl-name> in <psect>

Explanation: ERROR — More than one overlaid MAP area contains the same variable, but the variable's position differs in the MAP statements.

User Action: The same variable can appear in multiple overlaid MAPs, but the variable must occupy the same position in the PSECT; make sure that the variable appears in the same position in the MAP statements.

undefined line number

Explanation: ERROR — A statement tries to transfer control to a non-existent line.

User Action: Replace the non-existent line number with the correct destination line number.

undefined/unresolved global <name>

Explanation: FATAL — Either (1) a call was made to a subprogram that was not a loaded module or the current program module in memory when the RUN command was given, or (2) the compiler generated a global that does not exist in the RUN task.

User Action: (1) Make sure all external subprograms are either loaded or are the current program in memory. (2) If all subprograms are loaded or currently in memory and this message appears, submit an SPR and include all relevant information.

unexpected end of file

Explanation: ERROR — The compiler encountered an end-of-file immediately after an ampersand continuation character.

User Action: Remove the ampersand continuation character, or continue the line.

unresolved/undefined symbols

Explanation: ERROR — A program executed in the BASIC environment calls or invokes a subprogram or routine that has not been loaded with the LOAD command.

User Action: Load the subprogram or routine before running the program in the BASIC environment.

unsaved change has been made, CTRL/Z or EXIT to exit

Explanation: WARNING — A BASIC source program in memory has been modified, and an EXIT command or CTRL/Z has been typed. BASIC signals the error notifying you that if you exit from the compiler, the program modifications will be lost.

User Action: If you want to save the program, type the SAVE command. If you do not want to save the program, type EXIT or press CTRL/Z.

unterminated string literal

Explanation: ERROR — The program contains an improperly terminated string literal; for example, "ABC, "ABC', and 'ABC" are all improperly terminated.

User Action: Use the same type of quotation mark (either single or double) for both beginning and ending string delimiters.

user ABORT directive <text>

Explanation: FATAL — The compilation was terminated as the result of a %ABORT directive. The compiler prints the text following the %ABORT directive.

User Action: None.

user variable <name> not allowed in declaration

Explanation: ERROR — The parameter list in an external subprogram declaration contains a user variable name.

User Action: Remove the variable from the parameter list. When declaring a routine, the parameter list can contain only data type and parameter-passing mechanism specifications.

value too large for constant

Explanation: ERROR — The value of an EXTERNAL CONSTANT is larger than the specified data type allows.

User Action: Make sure the data type specified in the EXTERNAL CONSTANT statement matches that of the actual constant.

variable <name> not aligned in COMMON/MAP <name>

Explanation: WARNING — The total storage preceding a numeric variable in a COMMON or MAP is an odd number of bytes.

User Action: None. BASIC–PLUS–2 pads the preceding storage with a blank byte because the PDP–11 requires that numeric data start on a word boundary. VAX–11 BASIC does not pad the storage because numeric data can start on any byte boundary. However, if you want the program to run on both types of system, you should ensure that all numeric data start on a word boundary.

variable <name> not aligned in multiple references in MAP <name>

Explanation: ERROR — More than one overlaid MAP area contains the same variable, but the variable's position differs in the MAP statements.

User Action: The same variable can appear in multiple overlaid MAPs, but the variable must occupy the same position in the PSECT; make sure that the variable appears in the same position in the MAP statements.

variable or constant required

Explanation: ERROR — The program contains an executable DIM statement that contains an expression in the bounds list.

User Action: Remove the expression from the bounds list. Executable DIM statements can have only constants or variables (simple or subscripted) as bounds.

virtual array space exceeded at array <name>

Explanation: ERROR — The storage for virtual arrays on a single channel exceeds 2147483647 bytes.

User Action: If there is only one virtual array on the channel, you must reduce the amount of storage used by the array. However, if there is more than one virtual array on the channel, you can put each array on a separate channel.

virtual array string <name> length increased from <n> to <m>

Explanation: WARNING — In a string virtual array DIM statement, the specified string length is not a power of two.

User Action: None. BASIC increases the string length to the next higher power of two.

virtual array string <name> length truncated from <n> to <m>

Explanation: WARNING — A string virtual array specifies a string length greater than 512. BASIC truncates the length specification to 512.

User Action: None. The maximum string length for virtual arrays is 512.

WINDOWSIZE inconsistent with CLUSTERSIZE

Explanation: ERROR — An OPEN statement contains both a WINDOWSIZE and CLUSTERSIZE clause.

User Action: Remove either the WINDOWSIZE or the CLUSTERSIZE clause. CLUSTERSIZE is valid only on RSTS/E systems, and WINDOWSIZE is valid on all systems except RSTS/E.

Appendix B
Run-Time Error Messages

BASIC returns run-time error messages if an error occurs while a program is executing. There are three different types of error messages:

- Warning
- Error
- Fatal

Warning error messages indicate that an error has occurred, but program execution continues. In some cases, BASIC reprompts for more information or correct data; in other cases, BASIC performs the specified operation, but the results are not as expected. You do not need error-handling routines to trap errors that generate warning messages.

Error messages indicate that the program has aborted unless your program traps them in an error-handling routine. After handling the error, your program can continue.

Fatal error messages indicate that the program has aborted. You cannot trap fatal errors.

The following sections list run-time errors for PDP–11 BASIC–PLUS–2 in the following form:

<error number error message>

Explanation: WARNING, ERROR or FATAL — <text>

User Action: <text>

Section B.1 of this appendix lists PDP–11 BASIC–PLUS–2 run-time errors in numerical order, Section B.1.1 lists the run-time errors in alphabetical order, and Section B.1.2 lists error messages that PDP–11 BASIC–PLUS–2 does not generate but that can be displayed with the ERT$ function. Section B.2 lists PDP–11 BASIC–PLUS–2 debugger error messages. Debugger error messages are generated only if you compile program modules with the DEBUG qualifier.

B.1 PDP–11 BASIC–PLUS–2 Run-Time Errors

The PDP–11 BASIC–PLUS–2 error message format is:

<I><message> at line x in "module y"

where:

<I> Is a character indicating the severity of the error. The severity indicator can be:

% Indicating a warning

? Indicating an error or fatal error

<x> Is the line number where the error occurred.

<y> Is the name of the module where the error occurred.

The following section lists run-time errors (error number and message), their cause, and possible actions you can take to correct the error. Warning error messages are labeled WARNING. Trappable fatal error messages are labeled ERROR. Untrappable error messages are labeled FATAL.

Error Number **Error Message**

1 ?Bad directory for device

Explanation: ERROR—Either (1) the device directory does not exist or is unreadable, or (2) on RSTS/E systems, the program tried to access a magnetic tape with a file structure other than the default file structure.

User Action: Either (1) supply a valid directory, (2) change the default magnetic tape file structure with an ASSIGN MT0:.DOS or .ANSI command, or (3) override the default by specifying MODE 16384 for DOS format or MODE 24576 for ANSI format when you open the file.

2 ?Illegal file name

Explanation: ERROR—The file name (1) is too long, (2) is incorrectly formatted, (3) contains embedded blanks or invalid characters, or (4) on RSX–11M/M-PLUS systems, is in lowercase letters.

User Action: Supply a valid file specification.

3 ?Account or device in use

Explanation: ERROR—The specified operation cannot be performed because the device is already in use or because the account cannot be found or opened.

User Action: Wait until the device is available or specify another device or account.

4 ?No room for user on device

Explanation: ERROR—No user storage space exists on the specified device or on RSTS/E systems. You have exceeded the number of files allocated for your account.

User Action: Delete files that are no longer needed to free disk space in the account in which the error occurred.

5 ?Can't find file or account

Explanation: ERROR — The specified file or directory is not on that device.

User Action: Supply a valid file specification.

6 ?Not a valid device

Explanation: ERROR — The device is illegal or nonexistent.

User Action: Supply a valid device.

7 ?I/O channel already open

Explanation: ERROR — The program attempts to open an I/O channel that is already open for input or output.

User Action: Close the channel and reopen it or specify another channel.

8 ?Device not available

Explanation: ERROR — The requested device is in use.

User Action: Wait until the device is available or specify a different device.

9 ?I/O channel not open

Explanation: ERROR — The program attempted to perform an I/O operation before opening the channel.

User Action: Open the channel with the OPEN statement before attempting an I/O operation to it.

10 ?Protection violation

Explanation: ERROR — The program attempted to read or write to a file whose protection code did not allow the operation.

User Action: Use a different file, change the file's protection code, or change the attempted operation.

11 ?End of file on device

Explanation: ERROR — The program attempted to read data beyond the end of the file.

User Action: None. The program can trap this error in an error handler.

12 ?Fatal system I/O failure

Explanation: ERROR — An I/O error has occurred in either (1) the system or (2) Record Management Services. As a result, the last operation will not be completed.

User Action: See the *RMS–11 User's Guide* for information on RMS–11 errors, or retry the operation.

13 ?User data error on device

Explanation: ERROR — One or more characters may have transmitted incorrectly because of a parity error, bad punch combination on a card, or similar error.

User Action: Repeat the data entry operation.

14 ?Device hung or write locked

Explanation: ERROR — The program attempted an operation to a hardware device that is not functioning properly or is protected against writing.

User Action: Check the device on which the operation is performed.

15 ?Keyboard wait exhausted

Explanation: ERROR — No input was received during the execution of an INPUT, LINPUT, or INPUT LINE statement that was preceded by a WAIT statement.

User Action: None. You must supply input within the specified time.

16 ?Name or account now exists

Explanation: ERROR — The program attempted to create or rename a file or account with a file name that already exists.

User Action: Either (1) use the KILL statement to erase the old file before creating the file, or (2) use a different file name.

17 ?Too many open files on unit

Explanation: ERROR — The program attempted more than one DECtape output file per DECtape drive. BASIC permits only one open file per DECtape drive.

User Action: Close the file that is open on the unit or specify a different unit.

18 ?Illegal SYS() usage

Explanation: ERROR — RSTS/E only. The program attempted an illegal SYS call.

User Action: See the appropriate RSTS/E SYS call documentation.

19 ?Disk block is interlocked

Explanation: ERROR — RSTS/E only. The requested disk block segment is already in use (locked).

User Action: Try the operation again.

20 ?Pack IDs don't match

Explanation: ERROR — RSTS/E only. You have specified an incorrect identification code for the disk pack.

User Action: Use the correct pack ID.

21 ?Disk pack is not mounted

Explanation: ERROR — RSTS/E only. No disk pack is mounted on the specified disk drive.

User Action: Mount a disk pack on the disk drive or specify a disk drive that has a mounted disk pack.

22 ?Disk pack is locked out

Explanation: ERROR — RSTS/E only. The specified disk pack is mounted but is temporarily disabled.

User Action: Wait until the disk pack is available or specify another disk drive.

23 ?Illegal cluster size

Explanation: ERROR — RSTS/E only. The specified cluster size is unacceptable.

User Action: Change the cluster size. The cluster size must be a power of 2. A file cluster size must be equal to or greater than the pack cluster size and cannot be greater than 256. A pack cluster size must be equal to or greater than the device cluster size and cannot be greater than 16. The device cluster size is determined by the device type.

24 ?Disk pack is private

Explanation: ERROR — RSTS/E only. The program cannot access the specified disk pack.

User Action: Specify another disk pack.

25 ?Disk pack needs REBUILDing

Explanation: WARNING — RSTS/E only. The storage allocation table needs to be rebuilt and a nonfatal disk mounting error has occurred.

User Action: Use the CLEAN or ONLCLN operation in the UTILTY program.

26 ?Fatal disk pack mount error

Explanation: ERROR — RSTS/E only. The disk cannot be successfully mounted.

User Action: See your system manager.

27 ?I/O to detached keyboard

Explanation: ERROR — RSTS/E only. A program attempted to perform I/O to a hung-up dataset or to a detached console keyboard.

User Action: None. You cannot perform I/O to a detached keyboard or hung-up dataset.

28 ?Programmable ^C trap

Explanation: ERROR — A CTRL/C was pressed at the controlling terminal.

User Action: None. However, you can trap this error with an error handler.

29 ?Corrupted file structure

Explanation: ERROR — Either (1) RMS–11 has detected an invalid file structure on disk, or (2) on RSTS/E systems, a fatal error in a CLEAN operation has occurred.

User Action: See your system manager or the *RMS–11 User's Guide*.

30 ?Device not file-structured

Explanation: ERROR — A program attempted to access a nondisk device that is not file-structured. This error occurs, for example, when you try to gain a directory listing for a nondirectory device.

User Action: None. You cannot access a nondisk device that is not file-structured.

31 ?Illegal byte count for I/O

Explanation: ERROR — Either (1) The program contains a PUT statement with a COUNT value greater than the RECORDSIZE clause established in the OPEN statement or the buffer specified in the MAP statement, or (2) on RSTS/E systems, the disk is corrupted for your program if this error occurs when you try to execute the program.

User Action: Reduce the size of the COUNT clause to match the size of the buffer. You cannot put more characters in a buffer than the buffer's default or specified size.

32 ?No buffer space available

Explanation: ERROR — RSTS/E only. No buffer is available for file access. Possible causes are either (1) the receiving program has exceeded the pending message limit, or (2) the sending program has attempted to send a message and no small buffer is available for the operation.

User Action: See your system manager.

33 ?Odd address trap

Explanation: FATAL — Either (1) the program attempted to address nonexistent memory, or (2) on RSTS/E systems, the program attempted to access an odd address using the PEEK function.

User Action: None. Submit an SPR if this message appears for any reason other than those listed in the explanation and include all relevant output.

34 ?Reserved instruction trap

Explanation: FATAL — The program tried to execute an illegal or reserved instruction or a Floating-Point Processor (FPP) instruction on a system that does not have floating-point hardware.

User Action: None. If your system has floating-point hardware and this message appears, submit an SPR including all relevant information.

35 ?Memory management violation

Explanation: FATAL — Either (1) the program attempted to read or write to a memory location that does not allow access, or (2) on RSTS/E systems, attempted to access an address using the PEEK function.

User Action: None. Submit an SPR if this message appears for any reason other than those listed in the explanation and include all relevant output.

36 ?SP Stack Overflow

Explanation: ERROR — The program attempted to extend the program stack beyond its legal size.

User Action: None. If your program generates this message, submit an SPR and include all relevant information.

37 ?Disk error during swap

Explanation: ERROR — RSTS/E only. The system swapped your job into or out of memory. The contents of your job (the current task) are lost, but the job remains logged in to the system and control is returned to the keyboard monitor.

User Action: Report the occurrence of this error message to your system manager.

38 ?Memory parity failure

Explanation: ERROR — RSTS/E only. The memory occupied by your program has a parity error.

User Action: Contact your system manager.

40 ?Magtape record length error

Explanation: ERROR — RSTS/E only. A record in a file on magnetic tape was longer than the buffer designed to handle it.

User Action: Reduce the size of your record or increase the size of the buffer.

42 ?Virtual buffer too large

Explanation: ERROR — The program attempted to access a VIRTUAL file, and the buffer size was not a multiple of 512 bytes.

User Action: Change the I/O buffer to be a multiple of 512 bytes.

43 ?Virtual array not on disk

Explanation: ERROR — The program attempted to reference a virtual array on a nondisk device.

User Action: Virtual arrays must be on disk; change the file specification in the OPEN statement to open this array on disk.

44 ?Matrix or array too big

Explanation: ERROR — The program contains an array that is too large for memory.

User Action: Dimension the array with smaller subscripts.

45 ?Virtual array not yet open

Explanation: ERROR — The program attempted to reference a virtual array before the associated file on disk was opened.

User Action: Open the file on disk that contains the virtual array before you reference it.

46 ?Illegal I/O Channel

Explanation: ERROR — The program specified an I/O channel outside the legal range.

User Action: Specify I/O channels in the range 0 to 12, inclusive.

47 ?Line too long

Explanation: ERROR — The input line was longer than the record buffer.

User Action: Reduce the size of the input line to 255 characters.

48 %Floating point error

Explanation: WARNING — A program operation resulted in a floating-point number with an absolute value outside the range 10E–35 to 10E–38. If the program does not transfer to an error handling routine, BASIC returns a zero as the floating-point value for a number lower than 10E–38 and the system's maximum positive number for a number higher than 10E–38.

User Action: Check program logic or trap the error in an error handler.

50 %Data format error

Explanation: ERROR — The value supplied for a numeric variable is not a valid number, for example "ABC" and "1..2".

User Action: Supply numeric values of the correct data type.

51 %Integer error

Explanation: WARNING — The program requires an integer conversion from a larger data type (LONG or WORD) to a smaller data type (BYTE) and the resultant value is outside the allowable range. If the program does not transfer to an error handling routine, BASIC returns zero for the integer value.

User Action: Use an integer in the valid range. BYTE integers cannot be greater than 127; WORD integers cannot be greater than 32767; LONG integers cannot be greater than 2147483647.

52 ?Illegal number

Explanation: WARNING — The program specifies a data type in an INPUT or READ statement that does not agree with the value supplied. The number entered is too large for the desired variable.

User Action: Change the INPUT or READ statement or supply data of the correct type.

53 %Illegal argument in log

Explanation: WARNING — The program contains a negative or zero argument to the LOG or LOG10 function.

User Action: Supply an argument in the valid range.

54 %Imaginary square roots

Explanation: WARNING — An argument to the SQR function is negative. If the program does not transfer to an error handling routine, BASIC returns the square root of the absolute value of the argument.

User Action: Supply arguments to the SQR function that are greater than or equal to zero.

55 ?Subscript out of range

Explanation: ERROR — The program attempts to reference an array element outside of the array's dimensioned bounds.

User Action: Check program logic to make sure that all array references are to elements within the array boundaries.

56 ?Can't invert matrix

Explanation: ERROR — The program attempts to invert a single-dimensional array.

User Action: Supply a matrix array in the proper form for inversion.

57 ?Out of data

Explanation: ERROR — A READ statement requested additional data from an exhausted DATA list.

User Action: Remove the READ statement, reduce the number of variables in the READ statement, or supply more DATA items.

58 ?ON statement out of range

Explanation: ERROR — The index value in an ON GOTO or ON GOSUB statement is less than one or greater than the number of line numbers in the list.

User Action: Check program logic to make sure that the index value is greater than or equal to one and less than or equal to the number of line numbers in the ON GOTO or ON GOSUB statement.

59 ?Not enough data in record

Explanation: ERROR — You did not supply enough data to fill all the specified variables in an INPUT statement.

User Action: Supply enough data or reduce the number of specified variables.

60 ?Integer overflow, FOR loop

Explanation: ERROR — The value of the loop index in the program exceeds 32766%.

User Action: Make sure the loop index does not exceed 32766%.

61 %Division by 0

Explanation: WARNING — The program attempts to divide a value by zero. If the program does not transfer to an error handling routine, BASIC returns the value of zero.

User Action: Check program logic and change the attempted division, or trap the error in an error handler.

63 ?Field overflows buffer

Explanation: ERROR — A FIELD statement attempts to access more data than exists in the specified buffer.

User Action: Change the FIELD statement to match the buffer's size or increase the buffer's size.

64 ?Not a random access device

Explanation: ERROR — The program attempts a random access on a device that does not allow such access. This error occurs, for example, if you attempt a PUT operation with a RECORD clause to a file on magnetic tape.

User Action: Change the access to sequential instead of random or use a suitable I/O device.

65 ?Illegal MAGTAPE() usage

Explanation: ERROR — The program contains an incorrectly formatted or invalid MAGTAPE function code or function argument.

User Action: Change the MAGTAPE function code or argument.

72 ?RETURN without GOSUB

Explanation: FATAL — The program executes a RETURN statement before a GOSUB statement.

User Action: Check program logic to make sure that the RETURN statement is executed only in a subroutine or remove the RETURN statement.

73 ?FNEND without function call

Explanation: FATAL — The program attempts to execute an END DEF or FNEND statement before executing a function call.

User Action: Check program logic to make sure that the FNEND statement is executed only in a multi-line DEF or remove the END DEF or FNEND statement.

88 ?Arguments don't match

Explanation: FATAL — The arguments in a function call do not match the arguments defined for the function, either in number or in type.

User Action: Change the arguments in the function call to match those in the DEF statement or change the arguments in the DEF statement.

89 ?Too many arguments

Explanation: FATAL — A function invocation or CALL statement passed more arguments than were expected.

User Action: Reduce the number of arguments to the number expected. The maximum number of arguments is eight.

97 ?Too few arguments

Explanation: FATAL — A function invocation or CALL passed fewer arguments than were defined in the function or subprogram.

User Action: Change the number of arguments to match the number defined in the function or subprogram.

103 ?Program lost-Sorry

Explanation: FATAL — A fatal system error caused your program to be lost.

User Action: This error should never occur. Submit a Software Performance Report if this error occurs.

104 ?RESUME and no error

Explanation: FATAL — The program executes a RESUME statement outside of the error handling routine.

User Action: Check program logic to make sure that the RESUME statement is executed only in the error handler.

105 ?Redimensioned array

Explanation: FATAL — A matrix statement has tried to redimension an array larger than the array's initial allocation.

User Action: Do not redimension an array larger than its initial allocation. Correct the matrix statement.

116 ?PRINT-USING format error

Explanation: FATAL — The program contains a PRINT USING statement with an invalid format string.

User Action: Change the PRINT USING format string.

126 ?Maximum memory exceeded

Explanation: FATAL — The program has insufficient string and I/O buffer space because either (1) its allowable memory size has been exceeded, or (2) the system's maximum memory capacity has been reached.

User Action: Reduce the amount of string or I/O buffer space, or split the program into two or more modules.

127 %SCALE factor interlock

Explanation: WARNING — A subprogram was compiled with a different SCALE factor than the calling program.

User Action: Recompile one of the programs with a scale factor that matches the other.

128 ?Tape records not ANSI

Explanation: ERROR — The records on the magnetic tape you accessed are in neither ANSI D nor ANSI F format.

User Action: On RSX–11M/M–PLUS systems, remount the tape with the DOS (DO), Files–11 (RS) or RT–11 (RT) qualifier to determine the format of the records on the magnetic tape. On RSTS/E systems, set the magnetic tape to DOS format with the ASSIGN command or OPEN the file with the MODE 16384 (DOS) clause.

129 ?Tape BOT detected

Explanation: ERROR — The program attempts a rewind or backspace operation on a magnetic tape that is already at the beginning of the tape.

User Action: Check program logic; do not rewind or backspace if the magnetic tape is at its beginning.

130 ?Key not changeable

Explanation: ERROR — An UPDATE statement attempted to change a KEY field that did not have the CHANGES clause specified in the OPEN statement.

User Action: Specify the CHANGES clause for that key field in the OPEN statement. Note that the primary key cannot be changed and that you cannot specify the CHANGES clause when you open an existing file if the OPEN statement that created the file did not contain the CHANGES clause.

131 ?No current record

Explanation: ERROR — The program attempts a DELETE or UPDATE operation when the previous GET or FIND operation failed or when no previous GET or FIND operation was done.

User Action: Correct the cause of failure for the previous GET or FIND operation or make sure a GET or FIND operation was done, and then retry the operation.

132 ?Record has been deleted

Explanation: ERROR — A record previously located by its Record File Address (RFA) has been deleted.

User Action: None.

133 ?Illegal usage for device

Explanation: ERROR — The requested operation cannot be performed because:

- The device specification contains illegal syntax
- The specified device does not exist on your system

- The specified device is inappropriate for the requested operation (for example, trying to access an INDEXED file on a magnetic tape)

User Action: Supply the correct device type.

134 ?Duplicate key detected

Explanation: ERROR — In a PUT operation to an indexed file, a duplicate key was specified, and the DUPLICATES clause was not specified when the file was created.

User Action: Change the duplicate key or recreate the file specifying the DUPLICATES clause for that key.

135 ?Illegal usage

Explanation: ERROR — The program tried to open either a file of undeclared organization or a file without a record operation specified in the ACCESS clause.

User Action: Declare the file organization in the OPEN statement or specify the record operation you want to perform in the ACCESS clause.

136 ?Illegal or illogical access

Explanation: ERROR — The requested access is impossible because:

- The attempted record operation and the ACCESS clause in the OPEN statement are incompatible.
- The ACCESS clause is inconsistent with the file organization.
- The ACCESS READ or APPEND clause was specified when the file was created.

User Action: Change the ACCESS clause.

137 ?Illegal key attributes

Explanation: ERROR — The program specified an illegal combination of key characteristics.

User Action: Check the OPEN statement for either a NODUPLICATES clause and CHANGES clause or a CHANGES clause without a DUPLICATES clause.

138 ?File is locked

Explanation: ERROR — The program does not allow shared access and attempts to access a file that has been locked by another user or by the system.

User Action: Change the ACCESS or ALLOW clause in the OPEN statement to allow shared access or wait until the file is released by other user(s).

139 ?Invalid file options

Explanation: ERROR — The program has specified invalid file options in the OPEN statement.

User Action: Change the invalid file options.

140 ?Index not initialized

Explanation: ERROR — The program attempts a GET or FIND operation on a record in an empty INDEXED file.

User Action: None. You cannot perform GET or FIND operations on a record that does not exist.

141 ?Illegal operation

Explanation: ERROR — The program attempts to:

- Delete a record in a sequential file
- Update a record on a magnetic tape file
- Perform RMS–11 I/O on a virtual file (RSTS/E only)

User Action: Change the illegal operation. Block I/O requires virtual organization. RMS–11 I/O requires sequential, relative, or indexed organization.

142 ?Illegal record on file

Explanation: ERROR — A record contains an invalid record length.

User Action: Check the file for possible bad data.

143 ?Bad record identifier

Explanation: ERROR — The program attempted a record access that specified:

- A zero or negative record number on a RELATIVE file
- A GET or FIND operation on an INDEXED file with a null key

User Action: Change the record number or key specification to a valid value.

144 ?Invalid key of reference

Explanation: ERROR — The program attempted to perform a GET, FIND, or RESTORE operation on an INDEXED file using an invalid KEY clause, for example, an alternate KEY that has not been defined.

User Action: Use a valid KEY clause in the GET, FIND, or RESTORE statement.

145 ?Key size too large

Explanation: ERROR — The key length on a GET or FIND is either zero or larger than the key length defined for the target record.

User Action: Change the key specification in the GET or FIND statement.

146 ?Tape not ANSI labeled

Explanation: ERROR — The program attempts to access a file-structured magnetic tape that does not have an ANSI volume label.

User Action: Either (1) write an ANSI label when initializing the tape, or (2) change the access in the program to device-specific.

147 ?RECORD number exceeds maximum

Explanation: ERROR — The specified record number exceeds the maximum specified for this file or the maximum record number was negative when the file was created.

User Action: Reduce the specified record number.

148 ?Bad RECORDSIZE value on OPEN

Explanation: ERROR — Either (1) the value in the RECORDSIZE clause is zero or greater than 16384, or (2) the value does not match the RECORDSIZE clause used when the file was created.

User Action: Change the value in the RECORDSIZE clause.

149 ?Not at end of file

Explanation: ERROR — The program attempted a PUT operation either: (1) on a sequential file before the last record or (2) without opening the file with an ACCESS WRITE clause.

User Action: Open a sequential file with an ACCESS APPEND clause or open the file with an ACCESS WRITE clause.

150 ?No primary key specified

Explanation: ERROR — The program attempts to create an INDEXED file without specifying a PRIMARY KEY value.

User Action: Specify a PRIMARY KEY value.

151 ?Key field beyond end of record

Explanation: ERROR — The position given for the key field exceeds the maximum size of the record.

User Action: Specify a key field within the record.

152 ?Illogical record accessing

Explanation: ERROR — The program attempts to perform an operation that is invalid for the specified file organization, for example, a random access on a sequential file.

User Action: Supply a valid operation for that file organization or change the file organization.

153 ?Record already exists

Explanation: ERROR — An attempted random access PUT operation on a RELATIVE file has encountered a pre-existing record.

User Action: Specify a different record number in the RECORD clause of the PUT statement or delete the existing record.

154 ?Record/bucket locked

Explanation: ERROR — The program attempts to access a record or bucket that has been locked by another program.

User Action: Try the operation again.

155 ?Record not found

Explanation: ERROR — A random access GET or FIND operation was attempted on a deleted or nonexistent record.

User Action: None.

156 ?Size of record invalid

Explanation: ERROR — The program contains a COUNT clause specification that is invalid because COUNT:

- Equals zero
- Exceeds the maximum size of the record
- Conflicts with the actual size of the current record during a sequential file UPDATE operation on disk
- Does not equal the maximum record size for fixed format records

User Action: Supply a valid COUNT value.

157 ?Record on file too big

Explanation: ERROR — The specified record is longer than the record buffer.

User Action: Increase the record buffer's size.

158 ?Primary key out of sequence

Explanation: ERROR — RMS–11 has detected an error in a sequential PUT operation to an INDEXED file.

User Action: Change the PUT statement. If this does not work, the file is corrupted and you cannot do anything.

159 ?Key larger than record

Explanation: ERROR — The key specification exceeds the maximum record size.

User Action: Reduce the size of the key specification.

160 ?File attributes not matched

Explanation: ERROR — The following attributes in the OPEN statement do not match the corresponding attributes of the target file:

- ORGANIZATION
- BUCKETSIZE
- BLOCKSIZE
- KEY
- Record format

User Action: Change the OPEN statement attributes to match those of the file or remove the clause.

161 ?Move overflows buffer

Explanation: ERROR — The combined length of elements in the MOVE statement exceeds the record size defined for the file.

User Action: Reduce the size of the elements in the MOVE statement or increase the file's record size.

162 ?Cannot open file

Explanation: ERROR — The specified file cannot be opened.

User Action: Check the STATUS variable for system error codes.

164 ?Terminal format file required

Explanation: ERROR — The program attempted to use PRINT #, INPUT #, LINPUT #, MAT INPUT #, MAT PRINT #, or PRINT USING # to access a RELATIVE, INDEXED, or VIRTUAL file.

User Action: Supply a terminal-format file.

165 ?Cannot position to EOF

Explanation: ERROR — The operating system could not find the end of a sequential file opened with ACCESS APPEND. The file could be corrupted.

User Action: None.

166 ?Negative fill or string length

Explanation: ERROR — A MOVE statement contains a FILL item or string length with a negative value.

User Action: Change the FILL item or string length value to be greater than or equal to zero.

167 ?Illegal record format

Explanation: ERROR — The record format is illegal because:

- The specified record does not match the organization of the file.
- The specified record is illegal for the operating system on which the file resides.
- There are embedded carriage control characters in variable length records.

User Action: Correct the record format. Make sure the record has the same organization as specified when the file was created, that the record size is valid for the operating system, and that there are no embedded carriage control characters.

168 ?Illegal ALLOW clause

Explanation: ERROR — The value specified for the ALLOW clause is illegal for the type of file organization or for the operating system on which the file resides.

User Action: Change the ALLOW clause argument.

170 ?Index not fully optimized

Explanation: ERROR — A record was successfully written to an INDEXED file; however, the alternate key path was not optimized. This slows record access.

User Action: Delete the record and rewrite it.

171 ?RRV not fully updated

Explanation: ERROR — RMS–11 wrote a record successfully but did not update one or more Record Retrieval Vectors. Therefore, you cannot retrieve any records associated with those vectors.

User Action: Delete the record and rewrite it.

173 ?Invalid RFA field

Explanation: ERROR — During a FIND or GET operation by RFA, an invalid record's file address was contained in the RAB.

User Action: None. Please submit an SPR and include relevant output.

175 ?Bad node name

Explanation: FATAL — The specified node name is invalid, or, for NAME AS, the two node names are different.

User Action: Check node name.

180 ?No support for op in task

Explanation: FATAL — The program attempts to:

- Open a file that has an organization that is not specified by a qualifier to the BUILD command.
- Perform an I/O operation that requires RMS–11 file support not included in the BUILD command.

User Action: Change the ORGANIZATION clause in the OPEN statement or use the BUILD command with the correct qualifier.

182 ?Network operation rejected

Explanation: ERROR — A DECnet operation has failed.

User Action: Check that the DECnet link is available. Check that the operation is supported both over the network and by the remote node.

183 ?REMAP overflows buffer

Explanation: ERROR — The combined length of elements in the REMAP statement exceeds the record buffer defined for the file.

User Action: Reduce the size of the elements in the REMAP statement or increase the size of the record buffer specified by the MAP statement.

184 ?Unaligned REMAP variable

Explanation: ERROR — A REMAP statement attempts to put a WORD variable on an odd byte boundary.

User Action: Change the variables in the REMAP statement to align on an even byte boundary.

185 %RECORDSIZE overflows MAP

Explanation: WARNING — The file's record size specified in the RECORDSIZE clause is larger than the storage allocated by the MAP statement.

User Action: Reduce the file's record size with the RECORDSIZE clause or increase the amount of storage allocated by the MAP statement.

186 ?Improper error handling

Explanation: ERROR — The program attempts to execute a RESUME statement in a program module other than the parent module or the module where the error occurred.

User Action: Change the program logic to execute a RESUME statement in the parent module or the module where the error occurs.

243 ?CHAIN to non-existent line no

Explanation: ERROR — RSTS/E only. The program attempts to chain to another program using a line number that does not exist in the target program.

User Action: Change the line number to an existing line number in the target program.

246 ?Error trap needs RESUME

Explanation: ERROR — An error handler attempts to execute an END without first executing a RESUME statement.

User Action: Change the program logic so that the error handler executes a RESUME statement before executing an END statement.

247 ?Illegal RESUME to subroutine

Explanation: ERROR — While in an error handler activated by a subroutine, the error handler attempts to RESUME without a line number.

User Action: None. You cannot use RESUME without a line number if the current module name does not match the error module name; that is, you cannot RESUME to a subroutine unless you specify a line number.

248 ?Illegal return from subroutine

Explanation: ERROR — An external subroutine tries to execute a RETURN statement before the CALL statement calling the subroutine is executed.

User Action: Change the program so that the CALL statement comes before the RETURN statement.

250 ?Not implemented

Explanation: ERROR — The program attempted to use a language feature that does not exist in this version of BASIC, for example, TIME(4%).

User Action: Do not use the feature.

251 ?Recursive subroutine call

Explanation: ERROR — The program contains a subroutine that attempts to call itself.

User Action: A subroutine cannot call itself. Correct the program logic.

252 ?File ACP failure

Explanation: ERROR — The operating system's file handler reported an error to RMS–11.

User Action: The corresponding error value is stored in the STATUS variable. See the *RMS–11 User's Guide* for more information.

253 ?Directive error

Explanation: ERROR — A system service call resulted in an error.

User Action: The corresponding error value is stored in the STATUS variable. See the *RMS–11 User's Guide* for more information.

B.1.1 Alphabetical Listing of Run-Time Errors

Table B–1 lists the BASIC–PLUS–2 run-time errors (and their associated error numbers) in alphabetical order. This table is provided as a cross-reference; for a complete explanation of the errors and the appropriate user action, see Section B.1.

Table B–1: Alphabetical Listing of Run-Time Errors

Error Message	Error Number
?Account or device in use	3
?Arguments don't match	88
?Bad directory for device	1
?Bad node name	175
?Bad record identifier	143
?Bad RECORDSIZE value on OPEN	148
?Cannot open file	162
?Cannot position to EOF	165
?Can't find file or account	5
?Can't invert matrix	56
?CHAIN to non-existent line no	243
?Corrupted file structure	29
%Data format error	50
?Device hung or write locked	14
?Device not available	8
?Device not file-structured	30
?Directive error	253
?Disk block is interlocked	19
?Disk error during swap	37

(continued on next page)

Table B–1 (Cont.): Alphabetical Listing of Run-Time Errors

Error Message	Error Number
?Disk pack is locked out	22
?Disk pack is not mounted	21
?Disk pack is private	24
?Disk pack needs REBUILDing	25
%Division by 0	61
?Duplicate key detected	134
?End of file on device	11
?Error trap needs RESUME	246
?Fatal disk pack mount error	26
?Fatal system I/O failure	12
?Field overflows buffer	63
?File ACP failure	252
?File attributes not matched	160
?File is locked	138
%Floating point error	48
?FNEND without function call	73
?I/O channel already open	7
?I/O channel not open	9
?I/O to detached keyboard	27
?Illegal ALLOW clause	168
%Illegal argument in log	53
?Illegal byte count for I/O	31
?Illegal cluster size	23
?Illegal file name	2
?Illegal I/O Channel	46
?Illegal key attributes	137
?Illegal MAGTAPE() usage	65
?Illegal number	52
?Illegal operation	141
?Illegal or illogical access	136
?Illegal record format	167
?Illegal record on file	142
?Illegal RESUME to subroutine	247
?Illegal return from subroutine	248
?Illegal SYS() usage	18
?Illegal usage	135
?Illegal usage for device	133
?Illogical record accessing	152
%Imaginary square roots	54
?Improper error handling	186
?Index not fully optimized	170
?Index not initialized	140
%Integer error	51
?Integer overflow, FOR loop	60
?Invalid file options	139
?Invalid key of reference	144
?Invalid RFA field	173
?Key field beyond end of record	151
?Key larger than record	159
?Key not changeable	130

(continued on next page)

Table B–1 (Cont.): Alphabetical Listing of Run-Time Errors

Error Message	Error Number
?Key size too large	145
?Keyboard wait exhausted	15
?Line too long	47
?Magtape record length error	40
?Matrix or array too big	44
?Maximum memory exceeded	126
?Memory management violation	35
?Memory parity failure	38
?Move overflows buffer	161
?Name or account now exists	16
?Negative fill or string length	166
?Network operation rejected	182
?No buffer space available	32
?No current record	131
?No primary key specified	150
?No room for user on device	4
?No support for op in task	180
?Not a random access device	64
?Not a valid device	6
?Not at end of file	149
?Not enough data in record	59
?Not implemented	250
?Odd address trap	33
?ON statement out of range	58
?Out of data	57
?Pack IDs don't match	20
?Primary key out of sequence	158
?PRINT-USING format error	116
?Program lost-Sorry	103
?Programmable ^C trap	28
?Protection violation	10
?Record already exists	153
?Record has been deleted	132
?Record not found	155
?RECORD number exceeds maximum	147
?Record on file too big	157
?Record/bucket locked	154
%RECORDSIZE overflows MAP	185
?Recursive subroutine call	251
?Redimensioned array	105
?REMAP overflows buffer	183
?Reserved instruction trap	34
?RESUME and no error	104
?RETURN without GOSUB	72
?RRV not fully updated	171
%SCALE factor interlock	127
?Size of record invalid	156
?SP Stack Overflow	36
?Subscript out of range	55
?Tape BOT detected	129

(continued on next page)

Table B–1 (Cont.): Alphabetical Listing of Run-Time Errors

Error Message	Error Number
?Tape not ANSI labeled	146
?Tape records not ANSI	128
?Terminal format file required	164
?Too few arguments	97
?Too many arguments	89
?Too many open files on unit	17
?Unaligned REMAP variable	184
?User data error on device	13
?Virtual array not on disk	43
?Virtual array not yet open	45
?Virtual buffer too large	42

B.1.2 Errors Not Generated by PDP–11 BASIC–PLUS–2

The following errors cannot be generated in PDP–11 BASIC–PLUS–2. However, they can be displayed with the ERT$ function and are included for completeness.

Error Number	Error Message
39	?Magtape select error
41	?Non-res run-time system
49	%Argument too large in EXP
62	?No run-time system
66	?Missing special feature
67	?Illegal switch usage
68–70	Unused ERROR messages
71	?Statement not found
74	?Undefined function called
75	?Illegal symbol
76	?Illegal verb
77	?Illegal expression
78	?Illegal mode mixing
79	?Illegal IF statement
80	?Illegal conditional clause
81	?Illegal function name
82	?Illegal dummy variable
83	?Illegal FN redefinition
84	?Illegal line number(s)
85	?Modifier error
86	?Can't compile statement
87	?Expression too complicated
90	%Inconsistent function usage
91	?Illegal DEF nesting
92	?FOR without NEXT
93	?NEXT without FOR
94	?DEF without FNEND
95	?FNEND without DEF
96	?Literal string needed
98	?Syntax error

(continued on next page)

Error Number	Error Message
99	?String is needed
100	?Number is needed
101	?Data type error
102	?1 or 2 dimensions only
106	%Inconsistent subscript use
107	?ON statement needs GOTO
108	?End of statement not seen
109	?What
110	?Bad line number pair
111	?Not enough available memory
112	?Execute only file
113	?Please use the RUN command
114	?Can't CONTinue
115	?File exists-RENAME/REPLACE
117	?Matrix or array without DIM
118	?Bad number in PRINT-USING
119	?Illegal in immediate mode
120	?PRINT-USING buffer overflow
121	?Illegal statement
122	?Illegal FIELD variable
123	Stop
124	?Matrix dimension error
125	?Wrong math package
163	?No file name
169	?Unused ERROR message
172	?Record lock failed
174	?File expiration date unexpired
176–179	Unused ERROR messages
181	?Decimal overflow
187	?Illegal record locking clause
188–226	Unused ERROR messages
227	?String too long
228	?Record attributes not matched
229	?Differing use of /DOU
230	?No fields in image
231	?Illegal string image
232	?Null image
233	?Illegal numeric image
234	?Numeric image for string
235	?String image for numeric
236	?TIME limit exceeded
237	?1st arg to SEQ$ > 2nd
238	?Arrays must be same dimension
239	?Arrays must be square
240	?Cannot change array dimensions
241	?Floating overflow
242	?Floating underflow
244	?Exponentiation error
245	?Illegal exit from DEF
249	?Argument out of bounds
254–255	Unused ERROR messages